UNIVERSITÀ DI PISA

UNIVERSITÀ DEGLI STUDI DI PISA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea Specialistica in Tecnologie Informatiche

TESI DI LAUREA SPECIALISTICA

TECNICHE DI AGGREGAZIONE IN SISTEMI P2P: WAVELET E BLOOM FILTERS

CANDIDATO:

Matteo Mancini

RELATORI:

Prof.ssa Laura Ricci

Dott.ssa Barbara Guidi

CONTRORELATORE:

Prof. Stefano Chessa

ANNO ACCADEMICO 2012/2013

Indice

1 Introduzione **1**

2 Stato dell'arte **8**

2.1 Introduzione 8

2.2 Query complesse in sistemi P2P: problemi aperti 9

2.3 Classificazione sistemi P2P 12

2.3.1 Topologia della rete 12

2.3.2 Organizzazione delle strutture dati 14

2.4 Proposte in letteratura 15

2.4.1 MAAN 16

2.4.2 Squid 16

2.4.3 Baton 18

2.4.4 RST: Range Search Tree 22

2.4.5 Cone 26

2.4.6 Confronto soluzioni presenti in letteratura 32

3 Strumenti Matematici **34**

3.1 Wavelet: concetti generali 34

3.1.1 Cenni storici 34

3.1.2 La Trasformata Haar 35
3.1.3 Caratteristiche della trasformata di Haar 39
3.1.4 Trasformazione di Haar, livelli multipli 44
3.1.5 Haar wavelets 46
3.2 Wavelet: applicazioni 50
3.2.1 Wavelet e range query 50
3.3 Bloom Filters 62
3.3.1 Descrizione 62
3.3.2 Concetti matematici 63
3.3.3 Probabilità di falsi positivi 66
3.3.4 Utilizzo 68

4 Aggregazione di dati in HASP **70**
4.1 XCone 70
4.1.1 L'albero di mapping 70
4.1.2 La tabella di routing 73
4.1.3 Le operazioni 74
4.2 HASP 80
4.3 Aggregazione di chiavi derivate: strutture di digest 81
4.3.1 Aggregazione mediante Wavelet 82
4.3.2 Bloom Filters 96

5 Implementazione **98**
5.1 Framework Overlay Weaver 98
5.1.1 Architettura del framework 99
5.1.2 Tools di sviluppo 101

5.2 Definizione di uno scenario 103
5.3 Implementazione tecniche di digest 105
5.4 Codice degli algoritmi 107

6 Risultati sperimentali **113**
6.1 Valutazione wavelet 113
6.2 Tuning Bloom Filters 117
6.3 Confronto Wavelet vs Bloom Filters 120
6.3.1 Ambiente di test 120
6.3.2 Validazione Wavelet in PlanetLab 126
6.3.3 Risultati dei test 129

7 Conclusioni **136**
7.1 Sviluppi futuri 137

Elenco delle figure **142**

Bibliografia **147**

Capitolo 1

Introduzione

Le applicazioni distribuite basate sul modello P2P (Peer-to-Peer) [1] sono state negli ultimi anni oggetto di una grande diffusione nel settore informatico. La caratteristica fondamentale di tali sistemi è che tutti i nodi sono equivalenti, capaci di auto-organizzarsi e in grado di condividere un insieme di risorse nella rete. L'evoluzione delle applicazioni ha portato gli sviluppatori ad adottare architetture che si allontanassero sempre più dal modello client/server. Le reti P2P basano il proprio paradigma computazionale sulla decentralizzazione del controllo e sulla condivisione delle risorse; l'ambiente di esecuzione è estremamente dinamico e può assumere dimensioni di larga scala. Uno degli aspetti critici dei sistemi client/server è la presenza di un unico punto di fallimento e la limitata scalabilità su reti di larga scala. Le reti P2P non pretendono di sostituirsi a queste reti, ma piuttosto di fornire una valida alternativa in tutti quei contesti in cui le criticità del modello consolidato client/server rappresentano un problema.

Questa tesi si è focalizzata sulla definizione di supporto per il discovery di risorse in ambienti P2P che sia efficiente, immediato, scalabile ma soprattutto espressivo. In letteratura sono presenti diverse proposte, [10][11][12][13], focalizzate sul potenziamento dei modelli P2P al fine di incrementare l'espressività degli strumenti di ricerca delle risorse condivise. Risulta fondamentale che le proposte siano in grado di gestire efficacemente il dinamismo tipico dei

sistemi P2P senza introdurre elevati overhead di gestione e colli di bottiglia verso particolari peer della rete.

Gli obiettivi su cui si sta concentrando la ricerca riguardano il potenziamento dei modelli P2P al fine di migliorare ed aumentare l'espressività del linguaggio di query delle risorse sulla rete e la gestione dei vari tipi di attributi, in particolare è necessario esprimere correlazioni e/o combinazioni tra le proprietà delle risorse. Una prima classificazione delle query è basata sul numero di attributi in esse specificati e conduce alla distinzione tra:

- query monodimensionali, nel caso in cui il criterio di ricerca coinvolga un solo attributo;
- query multidimensionali, nel caso in cui il criterio di ricerca sia dato dall'unione dei vincoli su tutti o alcuni degli attributi che compongono le risorse.

Nel campo del discovery di risorse, particolare importanza rivestono le exact match query e le range query:

- **Exact match query**, in cui vengono specificati i valori esatti da ricercare per tutti gli attributi che compongono la risorsa. Ad esempio: *Trova delle risorse che abbiano* $OS = Linux$, $CPU = 2Ghz$, $RAM = 1GB$, $MemLibera = 512MB$ *e* $LOAD = 30\%$ *(Carico di lavoro).*
- **Range query**, in cui viene specificato per tutti gli attributi o solo per un sotto-insieme di essi, un range di valori ammissibili. Ad esempio: *Trova delle risorse che abbiano* $2Ghz \leq CPU \leq 3Ghz$ *e* $2GB \leq RAM \leq 4GB$ *e* $0\% \leq LOAD \leq 30\%$.

Si può dunque operare un'ulteriore classificazione su ciascuno degli attributi che caratterizzano una risorsa:

1. **Attributi statici**, la cui frequenza di aggiornamento rimane tendenzialmente costante nel tempo. Appartengono a questo tipo di categoria, attributi quali il sistema operativo o la frequenza di clock della CPU.

2. **Attributi dinamici**, i cui valori possono variare significativamente istante per istante. Esempi di questo tipo di attributi sono il carico di lavoro corrente sulla CPU, la memoria fisica disponibile o la percentuale di utilizzo della rete.

E' facilmente intuibile come le query multidimensionali risultino significativamente più complesse e diano origine a numerose problematiche. Tale complessità aumenta nel caso di attributi dinamici.

Lo studio di supporti per range query multiattributo è attualmente oggetto di ricerca. In particolare i classici metodi di ricerca su Distributed Hash Tables sono ormai consolidati, ma mentre supportano adeguatamente le exact match query, poco si adattano al supporto di range query. Infatti la funzione hash utilizzata nelle Distributed Hash Tables distrugge la località dei dati. Per questa ragione sono stati proposti molti approcci alternativi [15][16][17][18][19].

Una soluzione in grado di gestire il dinamismo delle risorse e che permetta di effettuare ricerche più espressive attraverso range query è *XCone*. Proposto in [31] ed ispirato a Cone [19], *XCone* integra la DHT con una struttura dati gerarchica il cui scopo è definire un albero logico di aggregazione delle chiavi distribuite sulla DHT che guidi la ricerca delle soluzioni di una range query unidimensionale. *XCone* eredita dalla DHT le proprietà di bilanciamento del carico tra i peer della rete in quanto non modifica la struttura della DHT stessa, ma utilizza un *trie* [21] costruito sugli identificativi della DHT, ma mantenuto in uno spazio distinto come struttura dati gerarchica aggiuntiva. Quando un peer si unisce alla rete *XCone* ottiene un identificatore dalla DHT e di conseguenza viene associato al rispettivo nodo foglia del trie per tutta la permanenza del peer in rete. In *XCone* le risorse sono memorizzate localmente nei peer che rappresentano le foglie dell'albero e ogni nodo logico interno dell'albero logico viene assegnato ad un peer attraverso una funzione di mapping. I descrittori delle risorse vengono aggregati secondo una funzione di aggregazione. La funzione di mapping di XCone determina quindi quali nodi logici non foglia vengono assegnati ad ogni peer della rete. La funzione di digest ha il compito di aggregare un insieme di valori in una informazione

di digest che richiede spazio di memoria limitato e che cattura nel migliore dei modi l'informazione sulla distribuzione delle chiavi. Una soluzione che estende *XCone* al caso di range query multiattributo è *HASP* [32]. *HASP* è basato sull'uso di curve space-filling [33][34][35] per la generazione di una chiave derivata a partire dal valore di n attributi che definiscono la singola risorsa. Le curve space-filling attraversano tutto lo spazio n-dimensionale delle risorse e permettono di identificare in modo univoco ciascuna n-upla di coordinate linearizzando uno spazio n-dimensionale e associando un valore univoco a ciascun punto di tale spazio. Tale valore prende il nome di *chiave surrogata* o *chiave derivata*. Lo spazio dei valori delle chiavi derivate risulta di dimensione notevole. Valori tipici per la classe di applicazioni considerata producono uno spazio di chiavi derivate la cui dimensione può essere pari a 2^{60}.

HASP utilizza come tecniche di digest i BitVectors Indexes [20] ed i Q-Digest [30]. La tecnica BitVector Indexes definisce un vettore binario di k bits in maniera tale da catturare la distribuzione dei valori di un attributo A definito in un intervallo $[a, b]$. Il numero k degli intervalli di separazione rappresenta il numero di elementi del BitVector in cui ogni elemento indica se almeno una chiave derivata appartiene a tale intervallo o meno. La tecnica Q-Digest sfrutta le proprietà matematiche dei quantili, per condensare una distribuzione di valori entro un margine di errore. Definito un insieme di valori n con dominio $[1, z]$, Q-Digest definisce un albero T di altezza $\log z$ in cui ogni livello partiziona l'intero dominio attraverso l'utilizzo di range $[min, max]$ assegnati ad ogni nodo. Data l'enorme ampiezza dello spazio delle chiavi abbiamo la necessità di comprimere l'albero T: definito un parametro di compressione k, è possibile eseguire l'operazione di compressione dell'albero raggruppando le chiavi verso i nodi di livello maggiore. Il parametro di compressione k influenza l'aggregazione dei nodi, all'aumentare di k vengono incrementati i valori da sintetizzare in un singolo nodo ciò conduce ad una approssimazione sempre maggiore.

La tecnica BitVector Indexes opera un'approssimazione troppo grossolana, mentre il Q-Digest riesce ad approssimare in modo più fine i dati contenuti

nelle foglie di HASP.

La presente tesi, dopo aver analizzato gli approcci proposti in letteratura, si è focalizzata sulla definizione di tecniche di digest più sofisticate da integrare in *HASP*. Le due nuove tecniche di digest proposte sono ispirate ai Bloom Filters [27] ed alle Wavelet [24][25] [26]. Il Bloom Filter è una struttura dati probabilistica compatta utilizzata per la rappresentazione di un insieme al fine di supportare queries di appartenenza. I Bloom Filters, a causa della loro rappresentazione compatta, possono restituire falsi positivi, cioè alcune queries potrebbero riconoscere in modo non corretto che un elemento appartiene ad un insieme; ma quando la probabilità di errore è mantenuta sotto una certa soglia permettono di ottenere trade-off interessanti tra lo spazio impiegato in memoria e la probabilità di falsi positivi. Un Bloom Filters rappresenta un insieme S di n elementi in un array di m elementi con $m \ll n$, a tale scopo utilizza k funzioni hash indipendenti che producono un valore distribuito uniformemente nel range $[1; m]$. Come vedremo nel capitolo 4 grazie alle loro proprietà matematiche i Bloom Filters possono essere creati con una percentuale di falsi positivi fissata, ovviamente maggiore accuratezza si richiede al Bloom Filters maggiore sarà lo spazio occupato in memoria. Durante l'integrazione dei Bloom Filters all'interno di HASP è stato svolto un lavoro di tuning per individuare il giusto trade-off tra la memoria utilizzata e l'affidabilità del Bloom Filters.

Le wavelet sono uno strumento matematico per la decomposizione gerarchica di funzioni. Esse vengono applicate in vari campi, nella teoria dei segnali, in geofisica, nei processi di denoising, etc. In particolare vengono usate per scomporre una data funzione $f(x)$ in funzioni più elementari che racchiudono informazioni riguardanti la funzione $f(x)$ stessa. A questo scopo le wavelet contengono delle informazioni che permettono la ricostruzione, a diversi livelli di dettaglio, della funzione originale. Esse offrono una tecnica elegante per la rappresentazione gerarchica di una funzione ottimizzando lo spazio utilizzato. Grazie alle wavelet è possibile costruire un istogramma gerarchico basato sulla distribuzione dei dati, in particolare sulle frequenze cumulate. Tale istogramma viene quindi utilizzato come funzione di digest per stima-

re l'ampiezza di range query. Come descritto sopra in *HASP* lo spazio dei valori delle chiavi derivate risulta di dimensione notevole ed inoltre le chiavi non sono distribuite in modo uniforme nel dominio. Allo scopo di utilizzare le wavelet per la creazione di istogrammi basati sulla distribuzione dei dati in uno spazio delle chiavi così grande è stato necessario introdurre una notazione ottimizzata, tale ottimizzazione ha portato all'introduzione di un nuovo algoritmo per il calcolo delle wavelet così come presentato nel capitolo 4. Per poter integrare le wavelet come informazione di digest nell'albero di aggregazione definito in *HASP* è stato necessario definire un algoritmo per il merge di due wavelet. Tale algoritmo sfrutta la possibilità di ricostruire i valori approssimati appartenenti ad entrambe le wavelet, una volta ottenuti tali valori calcola le nuove frequenze cumulate e di conseguenza ottiene una nuova wavelet che rappresenta entrambe le informazioni di digest.

I risultati sperimentali mostrano che l'utilizzo delle wavelet come tecnica di digest fornisce un'approssimazione più precisa delle informazioni contenute nei sotto-alberi dei nodi in HASP. Nella risoluzione di una query questo si traduce in una maggiore accuratezza del numero di risorse trovate rispetto al numero di risorse richieste. Inoltre la bontà dell'approssimazione restituita dalle wavelet conduce ad un minor traffico di rete all'interno di HASP.

L'elevata modularità nell'implementazione delle nuove tecniche di digest rende possibile modificare l'informazione di digest da utilizzare a seconda delle esigenze.

L'organizzazione della tesi è la seguente. Nel capitolo 2 sono presentati i principali approcci presenti in letteratura per la gestione di query complesse in reti P2P. Le varie proposte verranno analizzate e ne saranno evidenziati vantaggi e limiti. Nel capitolo 3 vengono presentate le nuove tecniche di digest analizzate in questa tesi dal punto di vista matematico mettendo in evidenza le proprietà che rendono tali tecniche un ottimo strumento da utilizzare per rappresentare il digest di una grossa mole di informazioni. Nel capitolo 4 verrano presentate le caratteristiche principali dell'architettura di HASP, saranno illustrate le funzioni di digest realizzate (Bloom FIlters e Wavelet) per l'impiego con chiavi derivate di grande dimensione. Nel capitolo 5 verrà

illustrata l'implementazione delle soluzioni proposte mettendo in evidenza gli algoritmi originali definiti in questa tesi. Nel capitolo 6 saranno descritti gli esperimenti per la valutazione delle tecniche di digest proposte ed i risultati sperimentali ottenuti.

Capitolo 2

Stato dell'arte

In questo capitolo verranno illustrati gli attuali approcci presenti in letteratura. In particolare verranno classificate le diverse proposte, evidenziandone i vantaggi ed i limiti.

2.1 Introduzione

Le reti P2P rappresentano attualmente una promettente soluzione per la condivisione di risorse in ambienti distribuiti. E' possibile notare come le Distributed Hash Table (DHT) siano alla base di numerosi servizi quali il file sharing, lo storage network ed il content delivery network. Negli ultimi anni si è inoltre assistito all'applicazione di tali sistemi verso nuovi contesti come il Gaming-online [3], il Semantic Web o il Grid Computing [2]. Visti i nuovi scenari la ricerca scientifica si è focalizzata su una ben nota problematica delle reti P2P: la mancanza di espressività nei criteri di selezione delle risorse. La capacità di recuperare in maniera efficiente un insieme di risorse che soddisfino dei criteri prefissati si rivela un requisito fondamentale sia nei sistemi di tipo Grid che nelle applicazioni di gaming online. Consideriamo il caso dei sistemi Grid, un sistema Grid fornisce un'infrastruttura di base indispensabile per la condivisione di un insieme di risorse, quali desktops, clusters computazionali, supercomputers, sensori o strumenti scientifici. Una

delle funzionalità di base, fornite dall'infrastruttura Grid, deve permettere all'utente di poter selezionare su larga scala un insieme di risorse che soddisfino i criteri impostati (Resource Discovery/Selection). In questi nuovi ambiti si sta affermando sempre di più l'utilizzo di reti P2P che sollevano nuovi interrogativi e problematiche da risolvere per la comunità scientifica.

2.2 Query complesse in sistemi P2P: problemi aperti

Attraverso l'utilizzo delle DHT, le reti Peer-to-Peer sono in grado di fornire un'infrastruttura scalabile e con funzionalità di base per la ricerca delle risorse. Le query, supportate da questi sistemi, sono del tipo: *Trova tutti i file il cui nome contenga una determinata stringa.* Se vogliamo utilizzare le reti P2P in nuovi contesti è necessario aumentare l'espressività del linguaggio di interrogazione, in particolare è necessario esprimere correlazioni e/o combinazioni tra le proprietà delle risorse.

Una prima classificazione delle query è basata sul numero di attributi in esse specificati e conduce alla distinzione tra:

1. Query monodimensionali o a singola dimensione, nel caso in cui il criterio di ricerca coinvolga un solo attributo.
2. Query multidimensionali, nel caso in cui il criterio di ricerca sia dato dall'unione dei vincoli su tutti o alcuni degli attributi che compongono le risorse.

E' facilmente intuibile come le query multidimensionali risultino significativamente più complesse e diano origine a numerose problematiche. Operando un'ulteriore classificazione, in base al criterio di selezione delle risorse sulla rete, possiamo suddividere l'insieme delle query multidimensionali in:

- **Exact match query**, in cui vengono specificati i valori esatti da ricercare per tutti gli attributi che compongono la risorsa. Ad esempio: *Trova delle risorse che abbiano* $OS = Linux$, $CPU = 2Ghz$,

$RAM = 1GB$, $MemLibera = 512MB$ *e* $LOAD = 30\%$ *(Carico di lavoro).*

- **Partial match query**, come le exact match query, ma in cui vengono specificati i valori esatti da ricercare solo per un sotto-insieme degli attributi che compongono la risorsa. Ad esempio, supponendo che una risorsa sia definita dai seguenti cinque attributi: *OS, CPU, RAM, memoria disponibile e carico di lavoro corrente (LOAD)*, la query risulterà del tipo *Trova delle risorse che abbiano* $OS = Linux$ *e* $CPU = 2Ghz$ *e* $RAM = 1GB$.

- **Range query**, in cui viene specificato per tutti gli attributi o solo per un sotto-insieme di essi, un range di valori ammissibili. Ad esempio: *Trova delle risorse che abbiano* $2Ghz \leq CPU \leq 3Ghz$ *e* $2GB \leq RAM \leq 4GB$ *e* $0\% \leq LOAD \leq 30\%$.

- **Boolean query**, in cui si specificano le condizioni su tutti o alcuni degli attributi attraverso l'utilizzo di operatori booleani. Ad esempio: *Trova delle risorse che (not* $RAM \leq 1GB$*) and (not* $OS = Linux$*).*

- **Nearest-Neighbor Query**, in cui vengono specificati per un insieme di attributi sia un insieme di condizioni che una funzione di valutazione delle risorse trovate definita metrica. La metrica consente di caratterizzare le risorse rispetto alla distanza dalla soluzione ottimale di ricerca. Ad esempio: *Trova delle risorse che abbiano* $CPU \geq 2Ghz$ *e* $1GB \leq RAM \leq 2GB$ *considerando come metrica la distanza dello spazio euclideo d-dimensionale.*

La classificazione delle query come sopra descritta è mostrata in Figura 2.1.

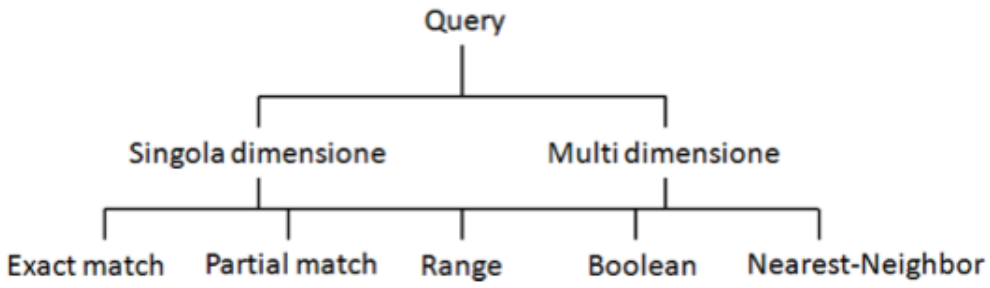


Figura 2.1: Classificazione query

Nella ricerca delle risorse che soddisfano dei determinati vincoli sul valore degli attributi che le compongono, è necessario tener conto anche della dinamicità delle risorse. Riprendendo l'esempio del Resource Discovery utilizzato in precedenza, una query del tipo *Trova k risorse con* $OS = Linux$ *e* $0\% \leq LOAD \leq 30\%$ e $CPU \geq 2Ghz$ *e* $1GB \leq MemoriaLibera \leq 4GB$, è composta da attributi, come il carico di lavoro corrente sulla CPU e la quantità di memoria fisica disponibile, i cui valori sono fortemente variabili nel tempo. Si può dunque operare un'ulteriore classificazione su ciascuno degli attributi che caratterizzano una risorsa:

1. **Attributi statici**, la cui frequenza di aggiornamento rimane tendenzialmente costante nel tempo. Appartengono a questo tipo di categoria, attributi quali il sistema operativo o la frequenza di clock della cpu.

2. **Attributi dinamici**, i cui valori possono variare significativamente istante per istante. Esempi di questo tipo di attributi sono il carico di lavoro corrente sulla cpu, la memoria fisica disponibile o la percentuale di utilizzo della rete.

Gli obiettivi su cui si sta concentrando la ricerca riguardano il potenziamento delle DHT al fine di migliorare ed aumentare l'espressività del linguaggio di query delle risorse sulla rete e la gestione dei vari tipi di attributi.

2.3 Classificazione sistemi P2P

L'architettura di una rete di tipo P2P risulta molto differente rispetto ad una rete classica di tipo client/server e cerca di essere quanto più possibile scalabile, tollerante ai guasti e completamente decentralizzata, sfruttando al massimo le potenzialità della rete di interconnessione Internet.

Possiamo analizzare le reti P2P secondo due aspetti:

1. La topologia della rete.
2. L'organizzazione delle strutture dati.

2.3.1 Topologia della rete

La topologia della rete indica la struttura mediante la quale i peers sono logicamente interconnessi e può essere di due tipi: *strutturata e non strutturata.*

Reti non strutturate. Una rete P2P non strutturata è solitamente descritta da un grafo in cui le connessioni dei peers sono basate sulla loro popolarità. Tra le varie reti P2P non strutturate, quali Gnutella [4], Napster [5], BitTorrent [6], JXTA [7] e Kazaa [8] possiamo distinguere vari livelli di decentralizzazione, dove, per livello di decentralizzazione, si identifica la possibilità di effettuare in maniera distribuita una ricerca e/o recupero delle risorse.

Una classificazione delle reti non strutturate può essere data dal tipo di ricerca che esse utilizzano: *deterministica* o *non deterministica.* Nel caso di ricerca deterministica, ciascuna risorsa, individuata da una query, viene localizzata in tempi certi; fanno parte di questa categoria di sistemi P2P la rete Napster e quella BitTorrent. Nel caso di ricerca non deterministica, reti P2P, quali Kazaa e Gnutella, mantengono un sistema di indici distribuito ed effettuano la ricerca delle risorse tramite un modello basato sul *flooding* dei messaggi ai vicini sulla rete. Ad esempio, in Kazaa, esistono due tipi di

peer: i peers normali e i super-peers. I super-peers raccolgono i dati ricevuti dai peers normali, eseguono un algoritmo di compressione delle informazioni ed eseguono il flooding dei messaggio di ricerca esclusivamente verso altri super-peers. Tali reti limitano il numero di messaggi scambiati sulla rete, ma risultano poco scalabili e presentano la problematica dei *falsi positivi* che riduce l'accuratezza delle operazioni di ricerca.

Reti strutturate. Appartengono alla categoria delle reti P2P strutturate, le reti CHORD [10], CAN [11], Pastry [12], Tapestry [13] e Kademlia [14] le quali utilizzano una struttura di base per la costruzione della rete. Nella maggior parte dei casi, la struttura utilizzata è la Distributed Hash Table (DHT), che garantisce un modello di ricerca di tipo deterministico e una complessità quasi sempre logaritmica nel numero di nodi presenti nella rete per le operazioni di ricerca e/o di inserzione. Le differenze che vi sono tra le reti P2P strutturate sopra citate, che utilizzano le DHT, sono da ricercarsi nei diversi algoritmi utilizzati per il posizionamento dei nodi nella rete, nell'assegnazione delle risorse ai nodi e nella ricerca di queste sulla rete DHT.

Uno schema riassuntivo delle reti P2P in cui viene evidenziata la distinzione tra reti strutturate e non strutturate, è visibile in Figura 2.2.

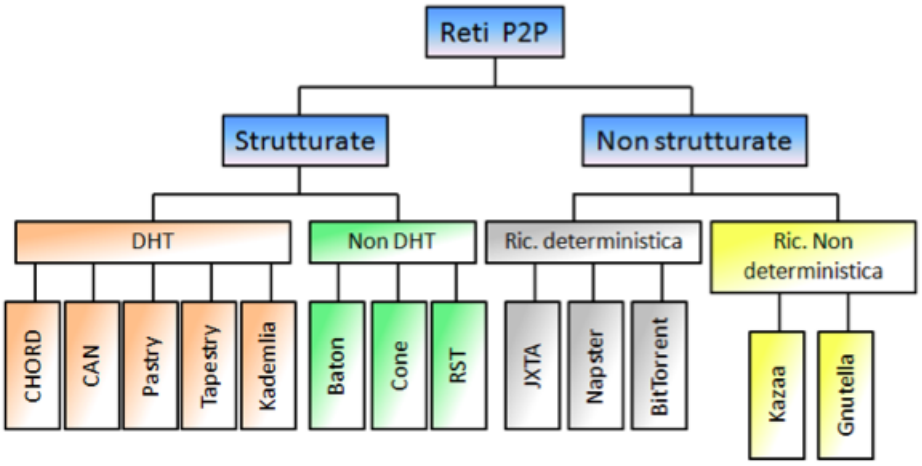


Figura 2.2: Topologia reti P2P

2.3.2 Organizzazione delle strutture dati

Le DHT costituiscono la struttura dati più utilizzata nelle reti P2P strutturate, in quanto garantiscono anche un supporto perfetto per l'esecuzione di query di tipo - *Exact Match* (vedi sezione 2.2). L'estensione degli attuali sistemi esistenti comporta necessariamente una ristrutturazione o potenziamento delle strutture dati attuali e, per fare ciò, in letteratura sono state individuate tre principali direzioni di ricerca che possono essere seguite:

1. utilizzo di tecniche di hashing con relative varianti;
2. utilizzo di curve di tipo space-filling;
3. utilizzo di strutture dati basate su alberi di ricerca;

Tecniche di hashing. Il primo approccio, basato sulla manipolazione delle funzioni hash, consiste nel cercare di preservare *alcune proprietà importanti come l'ordine o la località dei dati* in modo che la risoluzione delle query possa trarne vantaggio, sfruttando differenti euristiche studiate ad hoc. Tale approccio può dar luogo a criticità che devono essere opportunamente gestite in particolare nel caso di distribuzione non uniforme dei dati.

Curve space-filling. Il secondo approccio, basato sull'utilizzo di strutture dati, quali le curve *space-filling*, permette di effettuare un *mapping* da uno spazio n- dimensionale verso uno spazio uni-dimensionale o lineare. Una risorsa, definita da un insieme di n attributi, viene posizionata in uno spazio lineare tramite una specifica funzione di mapping differente a seconda della curva space-filling utilizzata. Il valore generato da tale funzione viene chiamato *chiave derivata* o *chiave surrogata*.

Alberi di ricerca distribuiti. Il terzo approccio è basato sull'utilizzo di strutture dati quali gli *alberi di ricerca distribuiti*, che possono essere suddivisi in due categorie distinte a seconda che si operi con dati lineari o dati multidimensionali. Nel caso di dati ad una dimensione, le strutture dati distribuite tendono a basarsi sull'utilizzo di *Hash Tree* o *B-alberi*, mentre nel caso di dati in uno spazio n-dimensionale si hanno strutture dati ispirate ad alberi

Space Driven come i *KD-Tree* o a strutture *Data Driven* quali gli *R-Tree*. In tutti questi casi le operazioni di ricerca non riportano falsi negativi.

Le proprietà auspicabili, che tutte le strutture dati dovrebbero possedere, sono due: il *bilanciamento del carico* in modo che ciascun nodo della rete in media mantenga una quantità di dati paragonabile e scambi un numero simile di messaggi con gli altri nodi e la *minimizzazione delle informazioni di stato*, in modo che ciascun nodo debba mantenere una quantità minima di informazioni per la gestione della struttura dati distribuita.

In Figura 2.3 è mostrato uno schema dell'organizzazione delle strutture dati nei sistemi P2P.

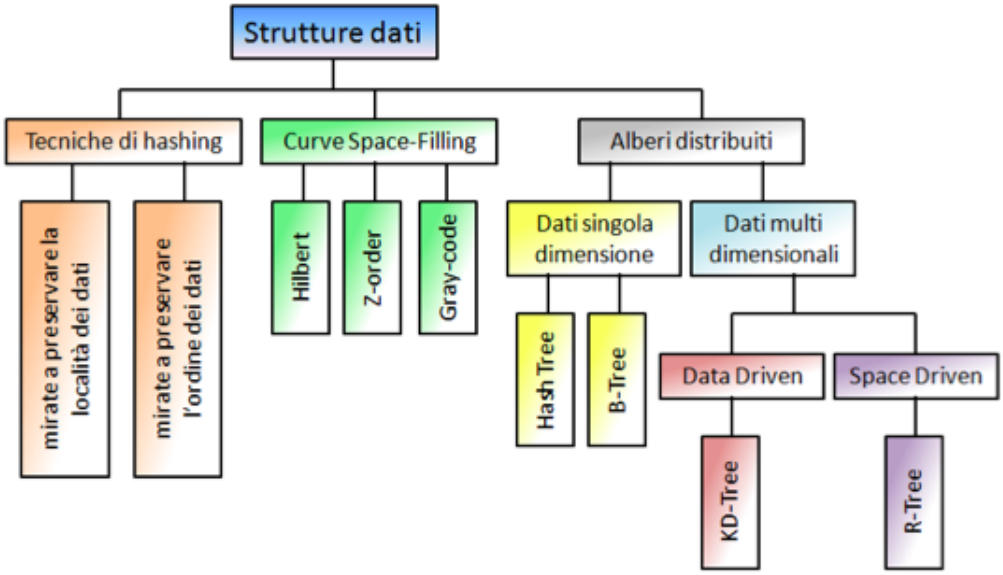


Figura 2.3: Organizzazione delle strutture dati P2P

2.4 Proposte in letteratura

In questa sezione sono illustrati gli approcci principali per ognuna delle direzioni di ricerca presentate nei paragrafi precedenti.

2.4.1 MAAN

MAAN [15] o Multi-Attribute Addressable Network è una rete P2P strutturata in grado di fornire supporto a range query multidimensionali. MAAN utilizza un anello Chord ed estende questa DHT tramite l'impiego di una funzione di hash in grado di preservare la località dei dati. Le risorse sono identificate da un insieme di coppie attributo-valore

$$(a_1 v_1), (a_2 v_2), \ldots, (a_n, v_n)$$

memorizzate secondo la funzione di hashing sull'anello Chord $[0 - 2^{m-1}]$. Il processo di memorizzazione sulla rete consiste quindi nell'applicare per ogni coppia la relativa funzione hash e nel memorizzare, nel corrispondente nodo della rete, l'intera risorsa. Da notare come una risorsa venga registrata un numero di volte pari al numero di attributi che la compongono. L'esecuzione di una range query multidimensionale in MAAN comporta la risoluzione di un insieme di query unidimensionali, una per ogni coppia attributo-valore della risorsa. Una volta ottenuti i risultati, la risoluzione della range query multidimensionale consiste in una semplice intersezione. Al fine di ottimizzare il processo appena descritto, gli autori propongono il concetto di attributo dominante. Un attributo è definito dominante se la percentuale di nodi filtrati risulta maggiore rispetto a quella degli altri attributi. Il nuovo processo di risoluzione di un'interrogazione consisterà nel risolvere esclusivamente la query unidimensionale sull'attributo dominante. Ricordando che ogni nodo memorizza l'intera risorsa, la ricerca per attributo dominante potrà risolvere localmente l'intera query multidimensionale, evitando l'invio di ulteriori messaggi.

2.4.2 Squid

Squid [16] è una rete P2P che utilizza la curva space-filling di Hilbert per memorizzare le risorse all'interno di un anello Chord e preservare la località dei dati. Squid permette dunque di effettuare range query multidimensio-

nali. Per stabilire l'allocazione di una risorsa sull'anello Chord, si calcola la sua chiave derivata tramite l'algoritmo di mapping della curva space-filling di Hilbert e si assegna la risorsa al primo nodo sull'anello che possiede un ID uguale o maggiore del valore della chiave surrogata appena calcolata. La funzione di Hilbert tramite un procedimento ricorsivo, identifica ad ogni iterazione un insieme di celle. I punti centrali di tali celle sono uniti dalla curva di Hilbert (Figura 2.4).

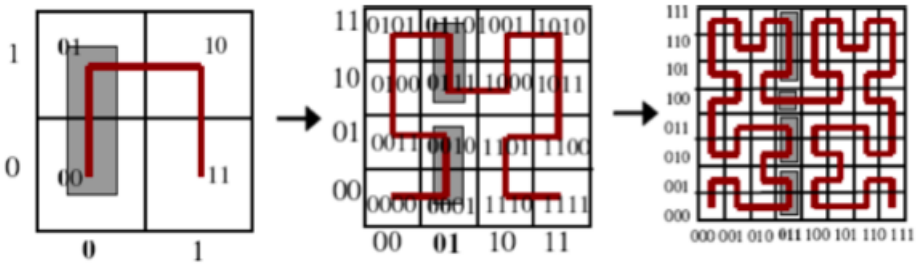


Figura 2.4: Squid: Funzione di Hilbert

Si definiscono *cluster* un insieme di celle generate ad una specifica iterazione della curva di Hilbert. In Squid l'elaborazione di una query prevede l'esecuzione di due passi distinti. Il primo passo prevede la traduzione della query in un insieme di *cluster rilevanti* e il secondo nell'interrogazione dei nodi corrispondenti (Figura 2.5). Un cluster è definito rilevante se almeno un punto al proprio interno appartiene all'insieme delle risorse mappate sull'anello Chord.

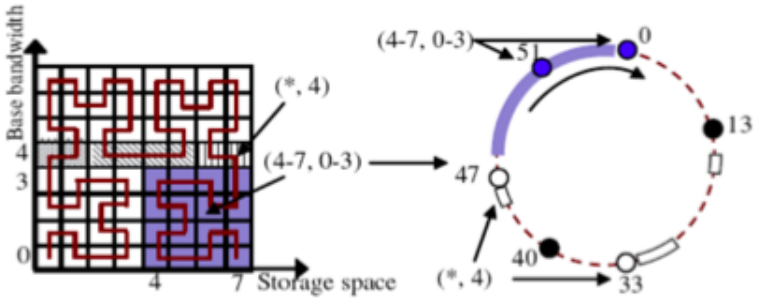


Figura 2.5: Squid clusters

In Squid, per evitare che un numero elevato di cluster possa rendere la soluzione non scalabile,è stato introdotto un ulteriore passo di ottimizzazione: la generazione dei cluster rilevanti avviene in maniera incrementale e guidata dai dati. Essendo ogni cluster identificato da un prefisso nell'albero, per poter ottimizzare il processo di ricerca, i cluster vengono raffinati incrementalmente dai soli nodi, che ricadono all'interno dei cluster via via raffinati (Figura 2.6).

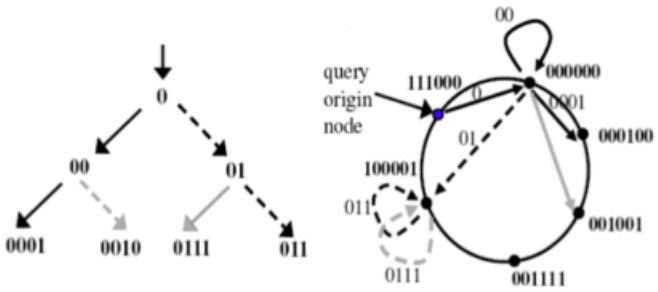


Figura 2.6: Raffinazione dei clusters in Squid

2.4.3 Baton

Baton, BAlanced Tree Overlay Network [17] è una rete P2P di tipo strutturato in grado di supportare range query monodimensionali mediante l'utilizzo di una struttura dati distribuita simile ad un *B-albero*. Grazie a tale struttura distribuita ad albero, Baton permette di effettuare query sia del tipo *Exact Match* che per *range*.

Ogni nodo dell'albero rappresenta esattamente un peer della rete e mantiene una parte della struttura dati distribuita. Ciascun nodo Baton è identificato tramite il proprio indirizzo IP e possiede come proprio stato interno, una routing table (Figura 2.7), che contiene il riferimento al nodo padre, i riferimenti ai nodi figli, ad un insieme di nodi adiacenti secondo una visita simmetrica o *in order* dell'albero e ad m nodi, vicini dello stesso livello. In particolare gli m nodi vicini a destra e a sinistra del nodo sono selezionati secondo la successione

$$m - 2^2, m - 2^1, m - 2^0, m + 2^0, m + 2^1, m + 2^2$$

supponendo una numerazione dei nodi per livelli successivi a partire dal livello zero e da sinistra verso destra.

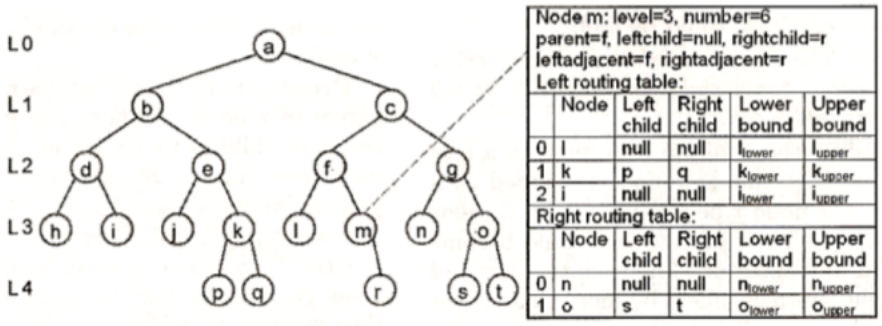


Figura 2.7: Routing table di un nodo in Baton

Baton, per poter effettuare delle ricerche, utilizza, oltre all'albero bilanciato, un'ulteriore struttura dati ad indici distribuita. Ciascun nodo si occupa di gestire un intervallo di valori, definito dagli estremi *lower* e *up*. L'assegnazione dell'intervallo di valori, che ogni nodo deve gestire, è stabilito con il seguente criterio: il valore *lower* assume il valore massimo contenuto nel sottoalbero sinistro del nodo, mentre *up* assume il valore minimo presente nel sottoalbero destro (Figura 2.8).

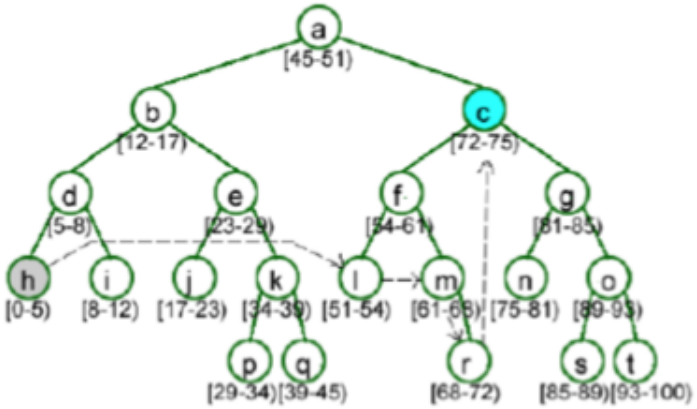


Figura 2.8: Indici distribuiti in Baton

In Figura 2.8 è possibile notare come l'assegnazione ai nodi foglia degli intervalli di valori da gestire sia perfettamente ordinata e come, a differenza dei B^+-alberi, anche i nodi interni dell'albero si occupino di gestire loro stessi un range di valori.

Attraverso l'esempio riportato in Figura 2.8 mostriamo adesso il processo di ricerca. Supponiamo che il nodo h voglia cercare un valore gestito dal nodo c. L'algoritmo di ricerca, illustrato in [17], effettua i seguenti passaggi. Il nodo h confronta il valore cercato con il proprio range, determinando, attraverso la sua routing table, che il valore di *up* risulta inferiore al valore cercato e dunque inoltra la query al vicino destro più estremo, tale per cui il relativo limite di *lower* risulti inferiore al valore cercato. Ricordiamo che i vicini sono memorizzati esponenzialmente sempre più distanti. Il nodo l effettua un procedimento analogo e inoltra la query al vicino m, il quale, non avendo nella propria routing table nessun vicino destro in grado di soddisfare la condizione di limite inferiore, invia la query al figlio destro r. Il nodo r, non trovando nessun vicino destro e non possedendo nessun figlio invia la query ad un nodo fisicamente adiacente, c, concludendo in questo modo l'operazione di ricerca.

Nel caso di range query, l'algoritmo si comporta in maniera del tutto analoga, intersecando i range esaminati. Dall'esempio proposto possiamo notare

come l'operazione di ricerca sia limitata nel numero di nodi da visitare. Infatti, come mostrato dagli autori, il costo di ricerca al caso pessimo risulta logaritmico e pari alla profondità dell'albero; inoltre si ha la certezza di non ottenere risultati falsi negativi.

L'operazione di *join* in Baton è separata in due fasi. Nella prima fase viene determinata la collocazione del nuovo nodo all'interno dell'albero e nella seconda vengono eseguite le operazioni di join vere e proprie. Il costo dell'operazione di join è determinato dalla prima fase di ricerca. Gli autori mostrano in [17] come tale ricerca non coinvolga più di $O(\log N)$ nodi, con N pari al numero di nodi o peers nella rete.

Mostriamo adesso come si comporta Baton nella gestione degli aggiornamenti dei valori ed in particolare di come possano innescare il bilanciamento dell'albero. Ipotizziamo che il valore gestito da un nodo abbia una variazione oltre i limiti del range gestito localmente e, considerando l'esempio in Figura 2.9, supponiamo che l'aggiornamento comporti lo spostamento di un valore dal generico nodo e al nodo g; questo spostamento può essere determinato attraverso una semplice procedura di ricerca. Supponiamo che il nodo g adesso si trovi a gestire un elevato numero di valori ed occorra procedere ad una suddivisione del range da lui gestito ed eventualmente ad un bilanciamento dei nodi. Il caso riportato in Figura 2.9 (a) mostra esattamente la situazione in cui è presente uno sbilanciamento del carico nel nodo g.

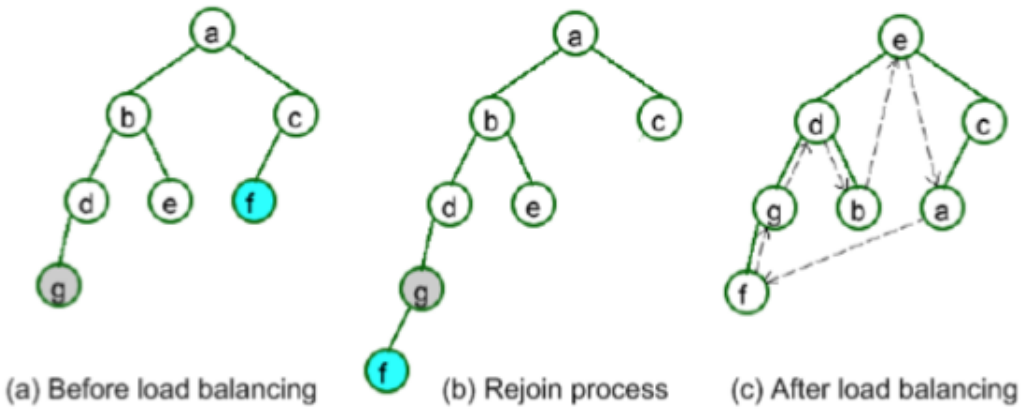


Figura 2.9: Esempilo di bilanciamento del cairco dei nodi in Baton

L'algoritmo di bilanciamento del carico procede nel seguente modo. Il nodo g identifica un nodo scarico presente in rete, per esempio il nodo f, il quale delega la gestione del proprio range di valori al nodo padre c ed effettua una nuova join sull'albero posizionandosi come nodo figlio di g; a questo punto il nodo g divide il proprio intervallo di valori da gestire con f (Figura 2.9 (b)). In seguito al bilanciamento del carico l'esempio in Figura mostra come sia necessario anche un bilanciamento dell'albero. La procedura è simile a quella utilizzata negli alberi AVL. I movimenti effettuati sono illustrati in Figura 2.9 (c) tramite la linea tratteggiata. Il nodo f rimpiazza il nodo g, il quale a sua volta rimpiazza il nodo d; il nodo d rimpiazza il nodo b, b rimpiazza il nodo e, e il nodo a ed infine a assume la posizione originale avuta da f.

Il problema del bilanciamento del carico si rivela il punto critico della proposta di Baton; infatti anche se il range di valori assegnati ad un nodo non varia, il numero di query ricevute da un nodo potrebbe essere signicativamente differente da quelle ricevute da altri nodi. Pertanto anche in questo caso il ribilanciamento dinamico della struttura è necessario. Si noti inoltre che tale bilanciamento del carico può comportare, nel caso peggiore, un trasferimento totale dei valori del nodo carico al nodo scarico e coinvolgere potenzialmente tutti i nodi dell'albero. Questa situazione potrebbe ricrearsi frequentemente in presenza di attributi altamente dinamici.

2.4.4 RST: Range Search Tree

La struttura dati *Range Search Tree (RST)* è la soluzione proposta in [18]. RST è una struttura basata sul mantenimento di un albero distribuito tra i peers della rete ed in grado di fornire supporto a range query multidimensionali. Per illustrare l'albero RST, occorre prima descrivere la tecnica di base utilizzata per effettuare il bilanciamento del carico tra i nodi. Una risorsa R è definita da un insieme di coppie attributo/valore della forma

$$R = \{(a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n)\}$$

Ad ogni coppia (a_i, v_i) è associata una matrice di bilanciamento del carico denominata *Load Balancing Matrix* o *LBM*. In questa matrice è tenuta traccia del carico relativo alle registrazioni e alle richieste effettuate per la coppia (a_i, v_i) ed in particolare serve ad organizzare i peers già presenti in rete, per condividerne il carico.

In Figura 2.10 è mostrata la matrice di bilanciamento per l'attributo (a_i, v_i). I nodi nelle colonne memorizzano una partizione della risorsa R, ovvero un sotto-insieme delle coppie attributo-valore, di cui è composta la risorsa con il vincolo che sia sempre inclusa la coppia (a_i, v_i). I nodi di una stessa colonna consistono invece di repliche della stessa partizione. La matrice *LBM* aumenta o diminuisce di dimensione dinamicamente nel tempo, a seconda del carico di registrazioni o interrogazioni effettuate nella rete. Per tenere traccia della dimensione della LBM si ricorre ad un *head node*, il quale memorizza il numero di partizioni e repliche.

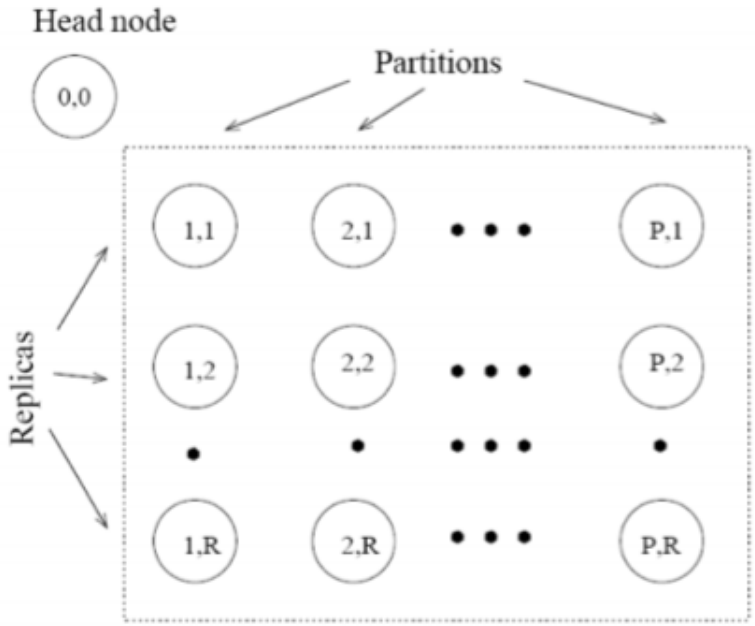


Figura 2.10: Matrice di bilanciamento del carico (LBM) per (a_i, v_i)

Descriviamo adesso come il generico nodo memorizzi una risorsa R attraverso

l'uso di tale matrice. Per ogni coppia di valori della risorsa, (a_i, v_i), vengono recuperate attraverso l'*head node* il numero di partizioni e repliche. In seguito tramite l'ausilio di soglie del carico, mantenute localmente, viene decisa la partizione p, in cui memorizzare la risorsa R e viene applicata la seguente funzione per determinare tutti i nodi replica r della stessa partizione:

$$N \leftarrow H((a_i, v_i), p, r)$$

La matrice LBM appena descritta risulta alla base della struttura Range Search Tree.

Un albero RST è costruito sul dominio dei valori di un singolo attributo. Tra i nodi di uno stesso livello si effettua il partizionamento del dominio in intervalli di differente granularità. Tutti i nodi di livello l_i possiedono degli intervalli di ampiezza minore rispetto ai nodi di livello $l_i + 1$, ma in ogni caso l'unione degli intervalli di un livello ricopre sempre l'intero dominio. Un esempio di partizionamento dell'albero è mostrato in Figura 2.11 (a); il dominio dei valori è definito nell'intervallo [0; 7].

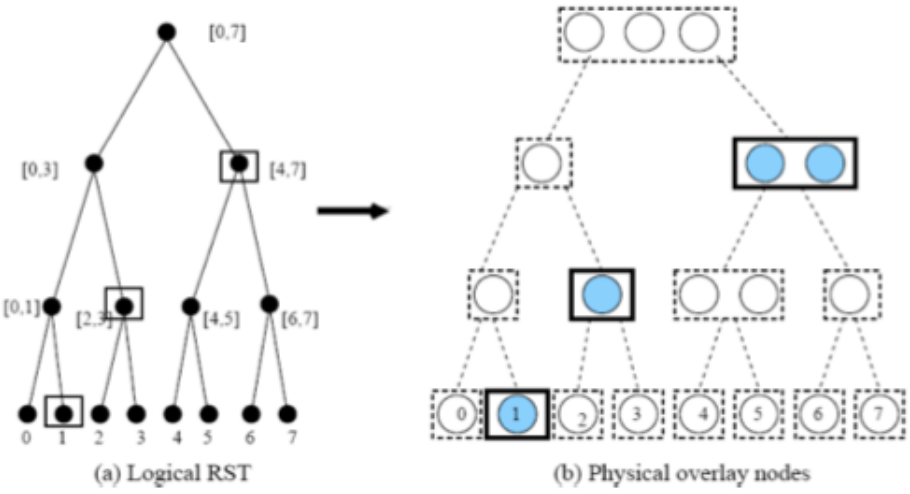


Figura 2.11: Esempio struttura RST

Illustriamo adesso le operazioni di registrazione ed interrogazione nell'albero RST. Per memorizzare una risorsa si estende l'algoritmo descritto precedentemente con la matrice LBM. Considerando la generica coppia (a_i, v_i), il valore v_i identifica un percorso specifico all'interno dell'albero RST denominato $Path(v_i)$. La memorizzazione della risorsa R viene effettuata esclusivamente tra i nodi del Path. Recuperando i valori della LBM, l'algoritmo viene poi esteso determinando i nodi nel seguente modo:

$$N \leftarrow H((a_i, v_i), [s, e], p, r)$$

i parametri $[s, e]$ rappresentano il range del nodo del path, in cui memorizzare i valori p ed r gli indici della relativa LBM (Figura 2.11(b)). Questo procedimento di memorizzazione, come notato degli autori, risulta non ottimizzato in quanto l'attributo potrebbe essere memorizzato in un sottoinsieme dei nodi del Path. Idealmente i nodi in cui memorizzare, dovrebbero essere i nodi RST i cui range siano non eccessivamente granulari, ma neanche tali da concentrare le registrazioni verso pochi nodi producendo uno sbilanciamento del carico. Per ottenere questo comportamento, viene introdotto il concetto di *banda* dell'albero RST. La banda rappresenta un sottoinsieme dei nodi di un Path, determinati dinamicamente dalla rete, in cui è conveniente memorizzare le risorse. Le informazioni sulla banda vengono mantenute in maniera simile a quanto fatto con la matrice LBM attraverso l'head node e il *Path Maintenance Protocol* (PMP) (Figura 2.12).

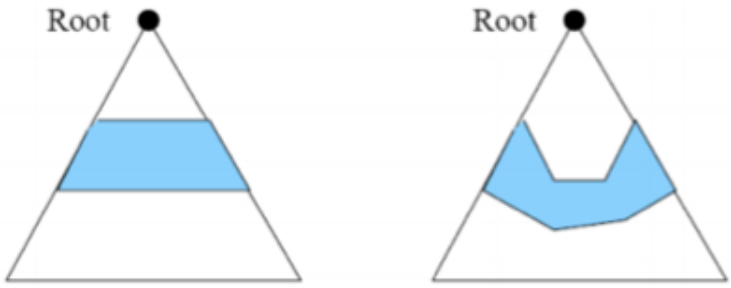


Figura 2.12: Determinazione della banda

Introducendo questa ottimizzazione, la procedura di ricerca varia leggermente, dovendo in via preliminare determinare la banda e successivamente memorizzare esclusivamente nei nodi del Path all'interno della banda.

Illustriamo adesso il meccanismo di interrogazione. Nella richiesta di una range query occorre notare come esistano diversi modi per decomporre il range richiesto utilizzando l'albero RST e l'efficienza dell'algoritmo di query è dunque in parte legata alla determinazione di una decomposizione ottimale della query e quindi dei nodi RST da contattare. A guidare il processo di decomposizione viene introdotta una metrica chiamata *rilevanza*. Supponendo di avere una range query Q con intervallo $[s, e]$, l'algoritmo di query potrà decomporre l'intervallo originario in k sub-queries corrispondenti a k nodi dell'albero RST, $N_1, N_{2,}, \ldots, N_k$. La *rilevanza* r è definita come

$$r = \frac{R_q}{\sum_{i=1}^{k} R_i}$$

dove R_i esprime l'ampiezza del range gestito dal nodo N_i e R_q l'ampiezza della query originaria. Intuitivamente la rilevanza indica quanto la decomposizione scelta corrisponda bene alla query originaria. Ulteriori dettagli sulla determinazione ottimale dei nodi RST possono essere trovati in [18]. A questo punto l'esecuzione di una range query si compone di tre passi: recupero preliminare delle informazioni sulla banda, decomposizione del range richiesto in una lista di nodi RST ed interrogazione dei vari nodi in accordo con la banda individuata e le informazioni contenute nella LBM.

2.4.5 Cone

Cone [19] è una rete P2P strutturata che utilizza una struttura dati ispirata ad un albero simile ad un heap. In Cone è possibile effettuare query del tipo *Trova k risorse di dimensione maggiore di S.* Cone è costruito sopra un livello di routing che supporti la ricerca basata sui prefissi, come ad esempio Chord. L'albero Cone è quindi un albero binario dei prefissi in cui le foglie vengono assegnate ai peers in maniera del tutto casuale attraverso la funzione

di hashing della DHT. Ogni nodo interno N possiede il valore massimo contenuto nel sottoalbero radicato in N, ottenuto mediante l'applicazione della *heap-property* (Figura 2.13). Un peer P mantiene una Routing Table, che memorizza una porzione consecutiva di nodi logici la quale corrisponde ad un percorso all'interno dell'albero. In particolare per ogni nodo logico N, non foglia, deve valere la seguente proprietà:

$$N = \begin{cases} left(N), & left(N).key < right(N).key \\ right(N), & altrimenti \end{cases}$$

Tale proprietà implica che il nodo N di livello $l > 0$ sia esso stesso anche uno dei nodi figli. Una volta collocato nell'albero Cone, il nodo non varierà la sua posizione durante tutta la permanenza in rete. Infatti la costruzione dell'albero Cone è determinata implicitamente dalle relazioni mantenute, distribuita tra tutti i nodi nelle *routing tables*. Una variazione del valore gestito comporterà una riorganizzazione delle tabelle di routing dei soli nodi coinvolti.

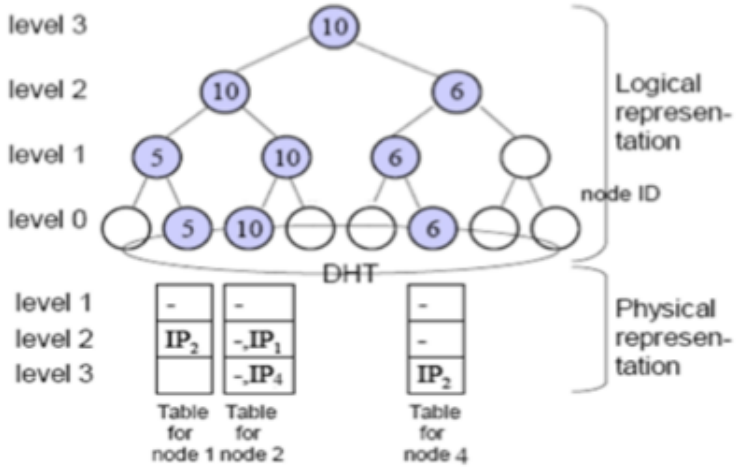


Figura 2.13: Architettura CONE

La memorizzazione di un valore avviene localmente e, come mostrato dagli autori, in un numero di nodi logaritmico. Per effettuare delle ricerche in Cone,

viene sfruttata la *heap-property*. La ricerca consiste in una risalita dell'albero attraverso l'utilizzo delle routing tables e sfrutta al massimo le operazioni di potatura dell'albero, per ridurne l'esplorazione. Il costo dell'operazione è logaritmico. Cone, pur essendo una struttura ad albero, presenta delle buone proprieta di bilanciamento del carico. Per poter analizzare la struttura occorre introdurre i concetti di *Data Traffic* e *Control Traffic*.

Data Traffic. Si definisce *Data Traffic* il carico di query che ogni nodo dell'albero gestisce. Dato un insieme di peers $P_1, P_2, \ldots, P_z$ aventi chiavi all'interno di un range, si definisce $Prob_D(P_i)$ la probabilità che il nodo P_i soddisfi una query. Il carico risulta bilanciato se vale

$$\forall i, j \leq z, i \neq j \qquad Prob_D(P_i) = Prob_D(P_j)$$

Control Traffic. Si definisce *Control Traffic* il numero di messaggi di controllo scambiati dai nodi. Idealmente si ha bilanciamento del carico se il numero di messaggi scambiati dai nodi di tutto il sistema è identico. Formalmente, definita $Prob_C(P_i)$ la probabilità che per qualche query un messaggio di controllo venga inviato al nodo N_i, si ha bilanciamento del carico se vale

$$\forall i, j \leq z, i \neq j \qquad Prob_C(P_i) = Prob_C(P_j)$$

Analisi del data traffic

Iniziamo considerando due esempi in cui si verifica il maggiore sbilanciamento del carico, per poi passare ad un'analisi più generale. Supponiamo di cercare una risorsa che soddisfi una determinata query. Ipotizziamo che tale query possa essere risolta esclusivamente da due peers P_1, P_2 e che l'associazione tra i peers e i nodi dell'albero Cone sia quella mostrata in Figura 2.14 (a). Il peer P_2 è il padre al nodo logico di livello uno del peer P_1. In questo caso si ha lo sbilanciamento massimo del carico e tutte le query originate dai T-1 peers saranno dirette a P_2 mentre a P_1 arriveranno le sole query originate da P_1 stesso. In questo caso lo sbilanciamento massimo è pari a

$O(T)$, con T pari al numero di peers presenti nella rete. Come caso degenere si può avere la situazione illustrata in Figura 2.14 (b), in cui i peers P_1 e P_2 abbiano ottenuto lo stesso identicativo dalla DHT e quindi siano stati assegnati alla stessa foglia. In questo caso P_1, ad esempio, rispondera' alle query originate da se stesso, mentre lascerà a P_2 la gestione di tutte le altre. Lo sbilanciamento anche in questo caso risulta $O(T)$. Fortunatamente la probabilità che avvenga una tale configurazione per entrambi gli scenari, risulta poco probabile. Infatti sia nel caso in cui i due nodi risultino adiacenti sia nel caso in cui abbiano generato lo stesso ID, la probabilità che questo accada è pari a $\frac{1}{2^h}$. Tale probabilità è calcolata assumendo che il numero di nodi fisici nella rete sia esattamente uguale al numero di foglie nell'albero; per valori inferiori di nodi la dimostrazione può essere effettuata in maniera analoga.

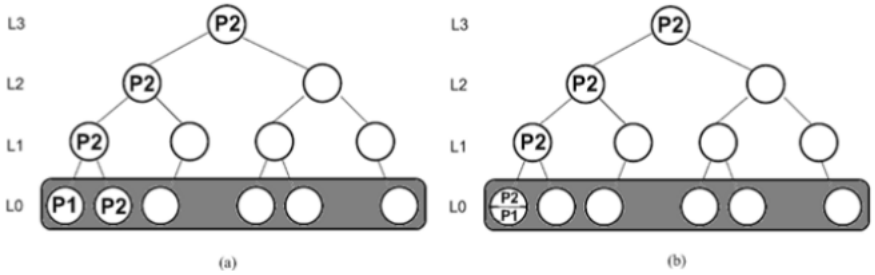


Figura 2.14: Casi di massimo sbilanciamento dell'albero Cone

A questo punto possiamo analizzare il caso medio. Consideriamo il caso in cui P_1 sia adiacente a P_2 attraverso un sotto-albero di altezza k e che P_2 sia il padre dal livello k fino alla radice (Figura 2.15).

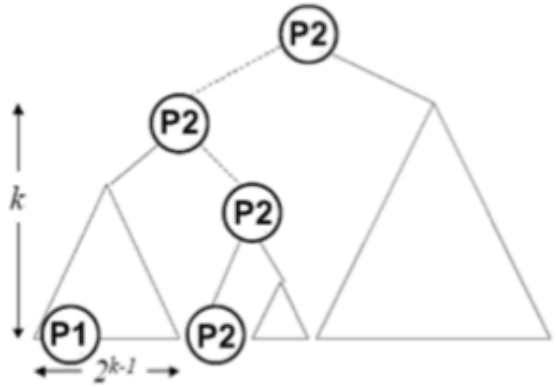


Figura 2.15: Caso generico del Data Traffic in Cone

L'assegnamento del relativo ID ai nodi è effettuato dalla funzione hash della DHT e la probabilità che P_1 sia esattamente in un sotto-albero adiacente di altezza k è pari a $\frac{2^k}{2^h}$.Pertanto la probabilità che due nodi P_1 e P_2 diventino adiacenti in un sotto-albero di altezza k è di $\frac{1}{2^{h-k}}$. Nello specifico P_2 diventerà il nodo padre di livello k. Tenendo conto di quanto detto possiamo definire le seguenti quantità:

- $Prob_D(P_1) = \frac{2^k}{2^h}$, la probabilità che il nodo P_1 soddisfi una determinata query.
- $Prob_D(P_2) = 1 - \frac{2^k}{2^h}$, la probabilità che il nodo P_2 soddisfi una determinata query.

Il rapporto tra i due Data Traffic risulta $r = \frac{Prob_D(P_1)}{Prob_D(P_2)} = \frac{2^h - 2^k}{2^k}$. Fissato k, sbilanciamento dell'albero risulta essere pari a

$$\left(\frac{1}{2^h - 2^k}\right) \times \left(\frac{2^h - 2^k}{2^k}\right) = \frac{2^h - 2^k}{2^h}$$

Quindi per calcolare lo sbilanciamento al caso generico occorre sommare per tutti i possibili valori di k:

$$\frac{2^h-1}{2^h}+\sum_{k=0}^{h-1}\frac{(2^h-2^k)}{2^h}=\frac{2^h-1}{2^h}+\frac{1}{2^h}\left(\left(2^h-1\right)+\left(2^h-2\right)+\ldots+\left(2^h-2^{h-1}\right)\right)=1-2^h+h+\sum_{k=1}^{n}\frac{1}{2^h}=h$$

Ottenendo quindi che nel caso medio, per la ricerca di una risorsa, il rapporto tra il peer selezionato il maggior numero di volte e quello selezionato il minor numero risulta non superiore a $h = \log(T)$.

Analisi del Control Traffic

Per poter effettuare l'analisi del Control Traffic è necessario fare la seguente ipotesi: la distribuzione delle chiavi possedute dai peers è identica alla distribuzione delle query effettuate, ed entrambe le distribuzioni sono continue. Questa ipotesi risulta necessaria per poter sfruttare le proprietà di simmetria e mettere in relazione la distribuzione delle chiavi con la distribuzione delle query. Se si considera l'esempio in Figura 2.15, la probabilità che una query originata dal nodo P_1 arrivi al nodo P_2, situato a livello k, è uguale alla probabilità che il valore massimo cercato dalla query sia il più grande valore tra tutti i valori gestiti nei 2^{k-1} peers del sotto-albero. Questo equivale a prelevare tra i $2^{k-1}+1$ campioni dalla distribuzione delle query (l'uno in piu' rappresenta la query) e stimare la probabilità che il valore richiesto sia esattamente il più grande valore tra tutti quelli gestiti nel sotto-albero. Dato che ogni elemento del campione, per ipotesi di simmetria, possa essere il valore massimo con uguale probabilità, allora la probabilità di selezionare il valore massimo risulta di $\frac{1}{2^{k-1}+1}$. Ritornando all'esempio in Figura 2.15, il numero di nodi dal quale la query può arrivare a P_2 è pari a 2^{k-1}. L'indice $k-1$ è dato dal fatto che in Cone, un peer che gestisce il nodo P_2, deve necessariamente gestire anche uno dei suoi figli, nell'esempio il figlio destro. Il carico complessivo di query che potrà raggiungere P_2 è quello proveniente dal solo sotto-albero di livello $k-1$; quindi i messaggi di controllo scambiati a seguito di un'interrogazione sono pari a $\left(2^{k-1}\right)\times\left(\frac{1}{2^{k-1}+1}\right)$. Possiamo concludere che al caso pessimo il fattore di massimo sbilanciamento del Control Traffic è rappresentato dal caso in cui P_2 sia situato nella radice dell'albero Cone, quindi:

$$\sum_{k=1}^{h} \left(\frac{2^{k-1}}{2^{k-1}+1} \right) < h = \log(T)$$

2.4.6 Confronto soluzioni presenti in letteratura

Le proposte illustrate presentano degli indubbi vantaggi in termini di aumento di espressività nel linguaggio d'interrogazione, ma occorre prestare attenzione agli elementi caratterizzanti e agli aspetti critici.

Nel caso delle soluzioni basate sulla manipolazione delle funzioni hash, come MAAN [15], la scelta della funzione hash risulta il fulcro centrale del modello prestazionale, che, nel caso di distribuzioni dei dati non uniformi, è estremamente critico. A questo aspetto si aggiunge l'ulteriore overhead che MAAN possiede nella presenza di risorse dinamiche, dovendo memorizzare tutti gli attributi della risorsa.

Nell'approccio presentato da Squid [16], le risorse sono state preventivamente trasformate da una funzione di mapping. Come al caso precedente, Squid, memorizzando le risorse direttamente sull'anello Chord, risulta essere sensibile alle distribuzioni non uniformi. In aggiunta si ha anche la presenza della nota problematica *curse of dimensionality*, che rende critica la gestione della struttura dati al crescere del numero di dimensioni dello spazio originario.

Infine negli ultimi approcci (Baton [17], RST [18] e Cone [19]) si è indagato su strutture basate sulla costruzione di una rete strutturata ispirata agli alberi. Queste soluzioni presentano degli aspetti fondamentali da analizzare:

- bilanciamento del carico;
- gestione degli attributi dinamici
- overhead di gestione

Nello specifico, l'approccio Baton mostra i suoi limiti sia con la presenza di attributi dinamici che con un elevato overhead di gestione in caso di bilanciamento della struttura; infatti in Baton la variazione di un attributo può

determinare uno sbilanciamento dei nodi e quindi innescare delle procedure di riorganizzazione della struttura.

Nella struttura RST i limiti maggiori si hanno sia nella scarsa scalabilità del sistema, basato nel recupero di informazioni essenziali attraverso *head node* che negli elevati costi di mantenimento dinamico delle LBM e banda.

L'ultimo approccio Cone risulta molto promettente, sia per il bilanciamento del carico sfruttando le proprietà di distribuzione dei dati attraverso funzione hash, che nel mantenimento della struttura a seguito dei cambiamenti dei valori. Per contro, Cone presenta una scarsa espressività sulle query effettuabili.

Lo studio della letteratura ha quindi mostrato come ogni proposta presenti soluzioni sub-ottime al problema del *Resource Discovery* diffcili da coniugare.

Capitolo 3

Strumenti Matematici

In questo capitolo verranno illustrati gli strumenti matematici utilizzati in questa tesi. In particolare verranno analizzate le Wavelets ed i Bloom Filters.

3.1 Wavelet: concetti generali

3.1.1 Cenni storici

Dagli inizi del 1900 fino al 1970 sono stati presentati diversi approcci alle wavelet: da una parte, la comunità matematica ha cercato di superare i limiti analitici della *Short Time Fourier Transform,* iniziando a concepire la trasformata wavelet continua; da un punto di vista sperimentale, invece, in diversi laboratori sono stati proposti algoritmi per l'analisi dei segnali o delle immagini attraverso, ad esempio, filtri in cascata che si rileveranno essere strettamente connessi con l'analisi wavelet.

Questi approcci, più o meno consapevoli, all'analisi wavelet partono nel 1909 con il lavoro del matematico tedesco A. Haar [22] che scopre l'omonimo sistema di basi ortonormali e proseguono con numerosi e diversi contributi.

Gli approcci basati su wavelet sono considerati, ad oggi, di grande interesse. Le molteplici caratteristiche delle wavelet le rendono estremamente duttili e funzionale in diverse discipline.

La tolleranza che la trasformata wavelet dimostra nei confronti di eventuali errori nei suoi coefficienti, lo rende strumento adatto per lo studio di risonanze magnetiche e per elettrocardiogrammi ad alta risoluzione.

Le wavelets trovano impiego anche nelle tecniche di compressione e riduzione del *rumore* in un segnale: infatti, la loro tolleranza a coefficienti errati consente l'azzeramento di parte dei coefficienti wavelet, quelli minori in modulo ad una certa soglia, consentendo così la rappresentazione dello stesso segnale con un numero inferiore di informazioni; allo stesso tempo, questo ne azzera la componente ad alta frequenza determinata dal *rumore*.

3.1.2 La Trasformata Haar

Le Haar wavelet [26], nella loro forma discreta, sono legate ad un'operazione matematica chiamata *Trasformata di Haar* la quale, inoltre, funge da prototipo per tutte le altre trasformate wavelet.

In questa sezione verranno introdotte le nozioni di base relative alla *Trasformata di Haar.*

La trasformata di Haar viene utilizzata principalmente per la rappresentazione di segnali, in particolare di segnali discreti.

Un segnale discreto è una funzione del tempo i cui valori corrispondono ad una serie di istanti discreti di tempo rappresentati nel dominio dei numeri interi. Un segnale discreto è rappresentato nella forma $f = (f_1, f_2, \ldots, f_N)$ dove N è un numero intero positivo che, nel formalismo delle trasformazioni di Haar, indica la *lunghezza* di *f.* I valori di f sono gli N numeri reali $f_1, f_2, \ldots, f_N$. Questi valori, tipicamente, vengono calcolati campionando un segnale analogico g agli istanti di tempo $t_1, t_2, \ldots, t_N$. Per cui, i valori di f risultano:

$$f_1 = g(t_1), f_2 = g(t_2), \ldots, f_N = g(t_N). \tag{3.1.1}$$

Per semplicità, assumeremo che gli incrementi di tempo che separano due valori successivi siano sempre gli stessi. Utilizziamo il termine *valori campionati* quando i valori di un segnale discreto sono definiti in questo modo.

Un esempio importante di *valori campionati* è l'insieme dei valori memorizzati in un file audio (.wav). Un altro esempio sono i valori di intensità audio registrati su un compact disc. Un ultimo esempio non riferito a segnali audio è quello di un elettrocardiogramma digitalizzato.

Come tutte le trasformate wavelet, la Trasformata di Haar decompone un segnale discreto in due sottosegnali di metà ampiezza. Un sottosegnale è chiamato trend, l'altro fluctuation.

Iniziamo analizzando il sottosegnale *trend.* Il primo sottosegnale *trend*, $a^1 = (a_1, a_2, \ldots, a_{N/2})$, del segnale *f*, è calcolato nel modo seguente. Il primo valore, a_1, è ottenuto eseguendo la media della prima coppia di valori di *f:* $(f_1+f_2)/2$ e moltiplicando tale valore per $\sqrt{2}$, per cui: $a_1 = (f_1 + f_2)/\sqrt{2}$. Allo stesso modo il valore successivo a_2 è calcolato eseguendo la media della coppia di valori successiva di *f:* $(f_3 + f_4)/2$, e moltiplicandola per $\sqrt{2}$. Ottenendo $a_2 = (f_3 + f_4)/\sqrt{2}$. Continuando con questo procedimento, tutti i valori di a^1 sono calcolati eseguendo le medie delle coppie di valori successivi e quindi moltiplicando il valore ottenuto per $\sqrt{2}$. La formula per ottenere un generico valore di a^1 è:

$$a_m = \frac{f_{2m-1} + f_{2m}}{2} \times \sqrt{2}$$

che è equivalente a

$$a_m = \frac{f_{2m-1} + f_{2m}}{\sqrt{2}} \tag{3.1.2}$$

per $m = 1, 2, \ldots, N/2$.

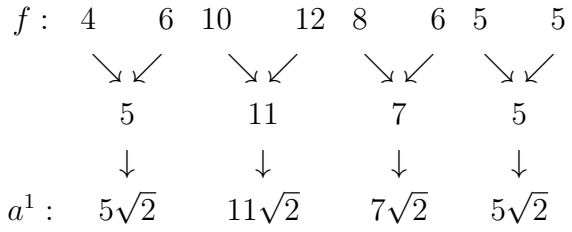
Ad esempio, supponiamo di avere una funzione f definita da otto valori:

$$f = (4, 6, 10, 12, 8, 6, 5, 5);$$

a questo punto calcoliamo il suo *trend:*

$$a^1 = (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 11\sqrt{2}).$$

Questo risultato può essere ottenuto utilizzando la formula sopra descritta o come indicato nel seguente diagramma:



A questo punto potrebbe non essere chiaro il motivo per cui sia necessaria la moltiplicazione per $\sqrt{2}$, nella prossima sezione vedremo come tale operazione sia essenziale per assicurare che la *Trasformata di Haar* conservi l'energia del segnale.

Analizziamo l'altro sottosegnale chiamato *fluctuation*. Il primo sottosegnale *fluctuation* del segnale *f*, rappresentato come $d^1 = (d_1, d_2, \ldots, d_{N/2})$ è calcolato nel modo seguente. Il primo valore, d_1, è ottenuto dalla metà della differenza della prima coppia di valori di f: $(f_1 - f_2)/2$, moltiplicato per $\sqrt{2}$. Per cui $d_1 = (f_1 - f_2)/\sqrt{2}$. Allo stesso modo, il secondo valore d_2, è calcolato prendendo la metà della differenza della seconda coppia di valori di *f:* $(f_3 - f_4)/2$ moltiplicato per $\sqrt{2}$. Abbiamo quindi $d_2 = (f_3 - f_4)/\sqrt{2}$. Proseguendo in questo modo, tutti i valori in d^1 sono il risultato di questa formula generale:

$$d_m = \frac{f_{2m-1} - f_{2m}}{\sqrt{2}} \tag{3.1.3}$$

per $m = 1, 2, \ldots, N/2$.

Ad esempio, prendendo in considerazione il segnale $f = (4, 6, 10, 12, 8, 6, 5, 5)$ definito sopra, il suo sottosegnale *fluctuation* d^1 risulta $(-\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$. Questo risultato puo' essere ottenuto utilizzando la formula sopra descritta o come indicato nel seguente diagramma:

$$
\begin{array}{lcccccccc}
f: & 4 & 6 & 10 & 12 & 8 & 6 & 5 & 5 \\
 & \searrow\swarrow & & \searrow\swarrow & & \searrow\swarrow & & \searrow\swarrow & \\
 & -1 & & 1 & & 1 & & 0 & \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\
d^1: & -\sqrt{2} & & -\sqrt{2} & & \sqrt{2} & & 0 &
\end{array}
$$

Trasformata di Haar, 1-livello

La trasformata di Haar viene eseguita in più fasi o livelli. Il primo livello è il *mapping* H_1 definito come:

$$f \overset{H_1}{\to} (a^1 \mid d^1) \tag{3.1.4}$$

da un segnale discreto f ai suoi sottosegnali *trend* a^1 e *fluctuation* d^1. Ad esempio, sopra abbiamo visto che:

$$(4,6,10,12,8,6,5,5) \overset{H_1}{\to} (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2} \mid -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0). \tag{3.1.5}$$

Il *mapping* H_1 ha un'inverso. L'inverso ricostruisce il segnale discreto f partendo dai due sottosegnali *trend* e *fluctuation* $(a^1 \mid d^1)$ attraverso la seguente formula:

$$f = \left(\frac{a_1 + d_1}{\sqrt{2}}, \frac{a_1 - d_1}{\sqrt{2}}, \ldots, \frac{a_{N/2} + d_{N/2}}{\sqrt{2}}, \frac{a_{N/2} - d_{N/2}}{\sqrt{2}} \right). \tag{3.1.6}$$

Infatti:

$$a_1 = \frac{f_1 + f_2}{\sqrt{2}}, \quad d_1 = \frac{f_1 - f_2}{\sqrt{2}}$$

da cui

$$a_1 + d_1 = \frac{f_1 + f_2 + f_1 - f_2}{\sqrt{2}} = \frac{2 \times f_1}{\sqrt{2}} = \sqrt{2} \times f_1 \Rightarrow f_1 = \frac{a_1 + d_1}{\sqrt{2}}$$

Analogamente:

$$a_1 - d_1 = \frac{f_1 + f_2 - f_1 + f_2}{\sqrt{2}} = \frac{2 \times f_2}{\sqrt{2}} = \sqrt{2} \times f_2 \Rightarrow f_2 = \frac{a_1 - d_1}{\sqrt{2}}.$$

Ad esempio, il seguente diagramma mostra come invertire la trasformata di primo livello descritta precedentemente:

$$\begin{array}{lcccccccc} a^1: & 5\sqrt{2} & & 11\sqrt{2} & & 7\sqrt{2} & & 5\sqrt{2} & \\ d^1: & -\sqrt{2} & & -\sqrt{2} & & \sqrt{2} & & 0 & \\ & \swarrow & \searrow & \swarrow & \searrow & \swarrow & \searrow & \swarrow & \searrow \\ f: & 4 & 6 & 10 & 12 & 8 & 6 & 5 & 5 \end{array}$$

3.1.3 Caratteristiche della trasformata di Haar

Consideriamo adesso quali sono i vantaggi nell'applicare la *Trasformata di Haar*. Questi vantaggi derivano dalla seguente proprietà.

Small Fluctuations. *I valori presenti nel sottosegnale fluctuation spesso hanno un valore di un ordine di grandezza significativamente più piccolo rispetto ai valori presenti nel segnale originale.*

La ragione per cui la proprietà Small Fluctuation è generalmente vera è che tipicamente abbiamo a che fare con segnali i cui valori sono campionamenti di segnali analogici continui g con un incremento del tempo tra i campionamenti molto piccolo.

In altre parole se, nel segnale discreto f, gli incrementi di tempo sono abbastanza piccoli, allora due valori successivi $f_{2m-1} = g(t_{2m-1)}$ e $f_{2m} = g(t_{2m})$ saranno vicini l'uno con l'altro a causa della continuità del segnale g. Conseguentemente, i valori della *Trasformata di Haar* soddisfano:

$$d_m = \frac{g(t_{2m-1}) - g(t_{2m})}{\sqrt{2}} \approx 0$$

Ad esempio, prendendo in esame il segnale $f = (4, 6, 10, 12, 8, 6, 5, 5)$ gli otto valori presenti nel segnale hanno un valore medio di 7. D'altra parte, il suo sottosegnale *fluctuation* $d^1 = (-\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$ ha un valore medio

di $0,75 \cdot \sqrt{2}$. In questo caso l'ordine di grandezza dei valori presenti nel sottosegnale *fluctuation* è di 6.6 volte minore rispetto a quello dei valori presenti nel segnale originale.

Vediamo un ulteriore esempio, consideriamo il segnale mostrato in Figura 3.1(a). Questo segnale è stato generato da 1024 campionamenti della funzione:

$$g(x) = 20x^2(1-x)^4 \cos 12\pi x$$

nell'intervallo $[0, 1)$. In Figura 3.1(b) è mostrato un grafico della *Trasformata di Haar* di primo livello. Il sottosegnle *trend* è mostrato nella prima metà del grafico, nell'intervallo $[0, 0, 5)$, il sottosegnale *fluctuation* è mostrato nella seconda metà del grafico, nell'intervallo $[0, 5, 1)$. Risulta evidente come una grande percentuale dei valori del sottosegnale *fluctuation* abbia un valore vicino allo zero, altro esempio della proprietà Small Fluctuations. Possiamo notare come il sottosegnale *trend* sia molto simile al segnale originale sebbene ridotto della metà in lunghezza ed esteso di un fattore $\sqrt{2}$ in verticale.

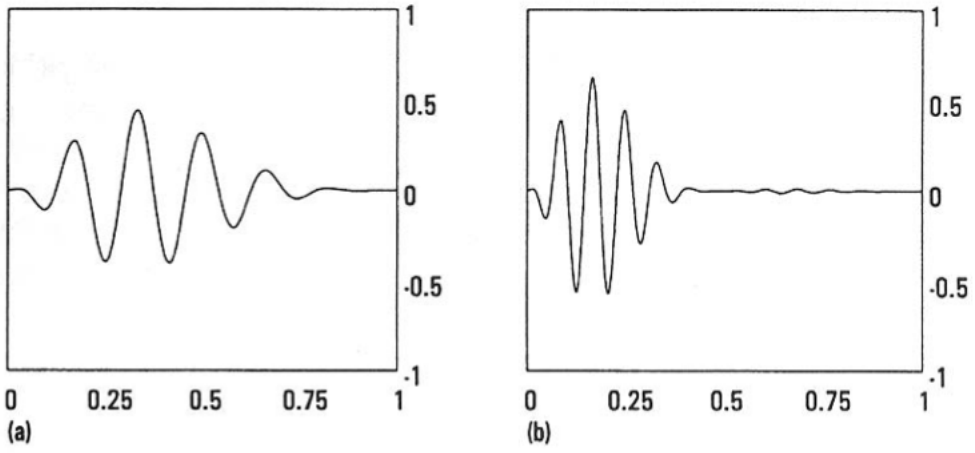


Figura 3.1: Trasformata di Haar su segnale discreto

Eseguendo un'analisi simile a quella della *fluctuation* possiamo capire perché il grafico del sottosegnale *trend* sia simile a quello del segnale originale. Se

g è continua e gli intervalli di tempo tra due campionamenti successivi sono molto piccoli, allora $g(t_{2m-1})$ e $g(t_{2m})$ avranno valori simili. Esprimendo questo fatto come approssimazione, $g(t_{2m-1}) \approx g(t_{2m})$, otteniamo la seguente approssimazione per ogni valore a_m del sottosegnale *trend:*

$$a_m \approx \sqrt{2}g(t_{2m}).$$

Questa equazione mostra come a^1 sia approssimativamente la stessa dei valori campione di $\sqrt{2}g(x)$ per $x = t_2, t_4, \ldots, t_N$. In altre parole, ci mostra che il grafico del sottosegnale *trend* ha un andamento simile al grafico di g, così come abbiamo visto nell'esempio precedente in relazione al segnale in Figura 3.1(a).

Uno dei motivi per cui la proprietà di Small Fluctuation è importante è perché ha diverse applicazioni nella *compressione dei segnali.* Comprimendo un segnale possiamo trasmettere i suoi valori, o un'approssimazione dei suoi valori, utilizzando un numero minore di bits. Ad esempio, potremmo inviare solamente il sottosegnale *trend* per il segnale mostrato in Figura 3.1(a) e quindi eseguire l'inverso della *Trasformata di Haar* considerando tutti i valori del sottosegnale *fluctuation* come se fossero zero, così da ottenere un'approssimazione del segnale originale. Dal momento in cui la lunghezza del sottosegnale *trend* è la metà rispetto a quella del segnale originale, abbiamo ottenuto una compressione del 50%.

Nella sezione precedente è stata definita la *Trasformazione di Haar* di primo livello.

Una volta eseguita la *Trasformata di Haar* di primo livello, è facile intuire come sia possibile ripetere questo processo per eseguire la *Trasformazioni Haar* multi-livello.

Un'importante proprietà della *Trasformata di Haar* è che essa *conserva l'energia del segnale.* Per *energia* di un segnale f si intende la somma dei quadrati dei suoi valori. Per cui, l'energia ε_f di un segnale f è definita da:

$$\varepsilon_f = f_1^2 + f_2^2 + \ldots + f_N^2. \tag{3.1.7}$$

Vediamo un esempio sul calcolo dell'energia. Supponiamo di avere $f = (4, 6, 10, 12, 8, 6, 5, 5)$, il segnale analizzato nella sezione precedente, ε_f è ottenuta come segue:

$$\varepsilon_f = 4^2 + 6^2 + \ldots + 5^2 = 446.$$

Perciò l'energia di f è 446. Inoltre, utilizzando i valori ottenuti dalla *Trasformazione di Haar* di primo livello $(a^1 \mid d^1) = (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2} \mid -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$, possiamo notare che:

$$\begin{aligned}\varepsilon_{(a^1|d^1)} =& a_1^2 + a_2^2 + \ldots + a_{N/2}^2 + d_1^2 + d_2^2 + \ldots + d_{N/2}^2 = \\ =& \frac{(f_1 + f_2)^2}{2} + \frac{(f_1 - f_2)^2}{2} + \ldots + \frac{(f_{N-1} + f_N)^2}{2} + \frac{(f_{N-1} - f_N)^2}{2} = \\ =& f_1^2 + f_2^2 + \ldots f_N^2\end{aligned}$$

per cui:

$$\varepsilon_{(a^1|d^1)} = 25 \cdot 2 + 121 \cdot 2 + \ldots + 2 + 0 = 446.$$

Quindi la *Trasformata di Haar* di primo livello ha mantenuto l'energia costante. Ciò è generalmente vero:

Conservazione dell'energia. *La Trasformata di Haar di primo livello conserva l'energia, $\varepsilon_{(a^1|d^1)} = \varepsilon_f$, per qualsiasi segnale f.*

Dall'esempio risulta chiaro il motivo per cui la moltiplicazione per $\sqrt{2}$ sia necessaria per il mantenimento dell'energia

La quantità ε_f è definita *energia* perché la somma dei quadrati viene utilizzata frequentemente in fisica quando viene calcolata una qualsiasi energia. Ad esempio, se una particella di massa ha velocità $v = (v_1, v_2, v_3)$, la sua energia cinetica è quindi $(m/2)(v_1^2 + v_2^2 + v_3^2)$. Quindi l'energia cinetica è proporzionale a $v_1^2 + v_2^2 + v_3^2 = \varepsilon_v$. Ignorando la costante di proporzionalità, $m/2$, otteniamo la quantità ε_v che chiamiamo energia di v.

La conservazione dell'energia è importante, ma è ancora più importante analizzare come la *Trasformata di Haar* ridistribuisca l'energia in un segnale comprimendo la maggior parte dell'energia nel sottosegnale *trend.* Ad esempio, per il segnale $f = (4, 6, 10, 12, 8, 6, 5, 5)$ nella sezione precedente abbiamo visto che il suo *trend* è $a^1 = (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2})$. Per cui, l'energia di a^1 è:

$$\varepsilon_{a^1} = 25 \cdot 2 + 121 \cdot 2 + 49 \cdot 2 + 25 \cdot 2 = 440.$$

D'altronde, il sottosegnale *fluctuation* è $d^1 = (-\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$, con un'energia di:

$$\varepsilon_{d^1} = 2 + 2 + 2 + 0 = 6.$$

Possiamo notare come l'energia del sottosegnale *trend* a^1 mantenga il $440/446 = 98,7\%$ dell'energia totale del segnale. In altri termini, la *Trasformata di Haar* di primo livello ha ridistribuito l'energia del segnale f in maniera tale che il 98% di essa fosse concentrata nel sottosegnale *trend* a^1 il quale è la metà di *f.* Questa caratteristica è chiamata *compressione dell'energia.* Come ulteriore esempio, consideriamo il segnale f mostrato in figura 3.1(a) e la sua *Trasformata di Haar* di primo livello mostrata in Figura 3.1(b). In questo caso, l'energia del segnale f è 127,308 mentre l'energia del sottosegnale *trend* a^1 è 127,305. Il 99,998% dell'energia totale è compattata nel sottosegnale a^1. Esaminando il grafico in Figura 3.1(b) è facile notare perché l'energia si sia compattata nel sottosegnale *trend;* i valori del sottosegnale *fluctation* d^1 sono molto piccoli in relazione ai valori del sottosegnale *trend,* cioè, l'energia ε_{d^1} contribuisce solo per una piccola frazione all'energia totale $\varepsilon_{a^1} + \varepsilon_{d^1}$.

Questi due esempi hanno mostrato il seguente principio generale:

Compressione dell'energia. *L'energia del sottosegnale trend* a^1 *mantiene una grande percentuale dell'energia del segnale trasformato* $(a^1 \mid d^1)$.

La proprietà di Compressione dell'energia è vera quando i valori del sottosegnale *fluctuation* hanno un ordine di grandezza più piccolo di quelli del sottosegnale *trend* (proprietà di Small Fluctuations).

3.1.4 Trasformazione di Haar, livelli multipli

Una volta eseguita la *Trasformata di Haar* di primo livello, è semplice ripetere lo stesso procedimento in modo da poter eseguire la *Trasformata di Haar* multilivello. Dopo aver applicato la *Trasformata di Haar* su un segnale f otteniamo il suo sottosegnale *trend* a^1 ed il suo sottosegnale *fluctuation* d^1. La *Trasformata di Haar* di secondo livello restituirà un secondo sottosegnale *trend* a^2 ed un secondo sottosegnale *fluctuation* d^2 operando solamente sul primo sottosegnale *trend* a^1.

Ad esempio, poniamo $f = (4, 6, 10, 12, 8, 6, 5, 5)$ come abbiamo visto sopra il suo sottosegnale *trend* di primo livello è $a^1 = (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2})$. Per ottenere i valori del secondo sottosegnale *trend* a^2, applichiamo la Formula 3.1.2 ai valori di a^1. Cioè, sommiamo coppie di valori successivi e le dividiamo per $\sqrt{2}$ come indicato nel seguente diagramma:

$$\begin{array}{lcccccc} a^1: & 5\sqrt{2} & & 11\sqrt{2} & & 7\sqrt{2} & & 5\sqrt{2} \\ & & \searrow\swarrow & & & & \searrow\swarrow & \\ a^2: & & 16 & & & & 12 & \end{array}$$

Per ottenere il secondo sottosegnale *fluctuation* d^2 di secondo livello applichiamo la Formula 3.1.3 ai valori di a^1. Cioè sottraiamo coppie di valori successivi di a^1 e li dividiamo per $\sqrt{2}$ come indicato nel seguente diagramma:

$$\begin{array}{lcccccc} a^1: & 5\sqrt{2} & & 11\sqrt{2} & & 7\sqrt{2} & & 5\sqrt{2} \\ & & \searrow\swarrow & & & & \searrow\swarrow & \\ d^2: & & -6 & & & & 2 & \end{array}$$

Abbiamo ottenuto così la *Trasformata di Haar* di secondo livello per il segnale *f:*

$$(a^2 \mid d^2 \mid d^1) = (16, 12 \mid -6, 2 \mid -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0).$$

Continuando ad applicare il procedimento visto sopra, possiamo ottenere la

Trasformata di Haar di terzo livello per il segnale *f:*

$$(a^3 \mid d^3 \mid d^2 \mid d^1) = (14\sqrt{2} \mid 2\sqrt{2}, -6, 2 \mid -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0).$$

Risulta interessante calcolare la compressione dell'energia avvenuta attraverso le *Trasformazioni di Haar* di secondo e terzo livello. Per il principio di Conservazione dell'energia sappiamo che $\varepsilon_{(a^2|d^2|d^1)} = 446$. Possiamo notare che $\varepsilon_{a^2} = 400$. Risulta perciò che la *Trasformazione di Haar* di secondo livello$(a^2 \mid d^2 \mid d^1)$ mantiene il 90% dell'energia totale contenuta nel segnale f in un sottosegnale *trend* a^2 che è $1/4$ della lunghezza di *f.* Questa è un ulteriore compressione, o *localizzazione,* dell'energia di *f.* Inoltre, $\varepsilon_{a^3} = 392$; perciò a^3 contiene l'87,89% del totale dell'energia di *f.* Questa è una compressione ancora maggiore; la *Trasformata di Haar* di terzo livello $(a^3 \mid d^3 \mid d^2 \mid d^1)$ contiene almeno l'88% dell'energia totale contenuta in f con un segnale di lunghezza $1/8$ rispetto a quello originale.

Come ulteriore esempio di come la *Trasformata di Haar* ridistribuisce e localizza l'energia in un segnale, consideriamo il grafico mostrato in Figura 3.2. In Figura 3.2(a) è mostrato il segnale originale, e in Figura 3.2(b) è mostrata la *Trasformata di Haar* del segnale. Nelle Figure 3.2(c) e 3.2(d) vediamo la localizzazione dell'energia dei rispettivi segnali. Come possiamo notare nella Figura 3.2, la *Trasformata di Haar* di secondo livello ha ridistribuito e localizzato l'energia del segnale originale.

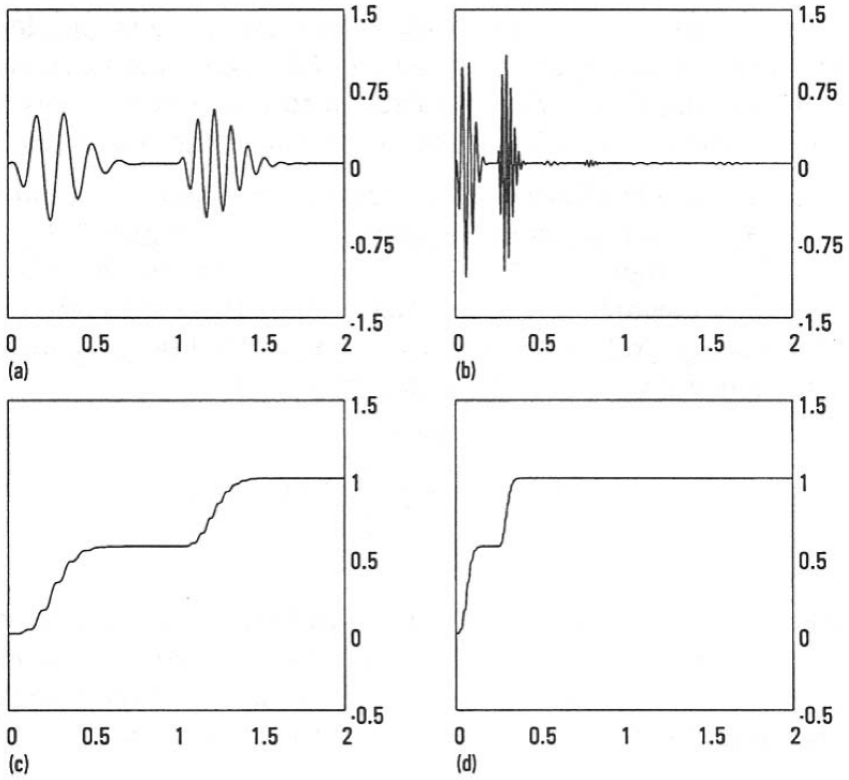


Figura 3.2: (a) Segnale originale. (b) Trasformata di Haar di secondo livello. (c) Energia del segnale originale. (d) Energia del segnale risultante dalla Trasformata di Haar di secondo livello.

3.1.5 Haar wavelets

Le Haar wavelets sono tra i più semplici tipi di wavelets, sono definite da una base ortonormale il cui prodotto con qualsiasi segnale f produce il sottosegnale *fluctuation* d^1 di f. In particolare quelle di primo livello sono definite come:

$$
\begin{aligned}
W_1^1 &= \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, \ldots, 0\right) \\
W_2^1 &= \left(0, 0, \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, \ldots, 0\right) \\
&\vdots \\
W_{N/2}^1 &= \left(0, 0, \ldots, 0, \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)
\end{aligned}
$$

Le Haar wavelets di primo livello hanno diverse proprietà interessanti. Per primo, ciascuna di esse ha un energia pari a 1. Secondo, ognuna consiste di una rapida fluttuazione tra i due soli valori diversi da zero, $\pm 1/\sqrt{2}$. Da qui il nome *wavelet.* Infine, sono tutte molto simili tra loro in quanto ognuna di esse è una traslazione di un numero pari di unità della prima Haar wavelet W_1^1. La seconda Haar wavelet W_2^1 è una traslazione in avanti di due unità rispetto a W_1^1; così come W_3^1 è una traslazione in avanti di quattro unità rispetto a W_1^1, e così via.

Le Haar wavelets di primo livello riescono ad esprimere il sottosegnale *fluctuation* d^1 in maniera semplice facendo uso del prodotto scalare con queste wavelets.

Ricordiamo che il prodotto scalare tra due segnali è un'operazione elementare, ed è definito come:

$$f \cdot g = (f_1 g_1 + f_2 g_2 + \ldots f_N g_N) \tag{3.1.8}$$

Utilizzando le Haar wavelets di primo livello, possiamo esprimere i valori del primo sottosegnale *fluctuation* d^1 come prodotto scalare. Ad esempio:

$$d_1 = \frac{f_1 - f_2}{\sqrt{2}} = f \cdot W_1^1.$$

Allo stesso modo abbiamo $d_2 = f \cdot W_2^1$, e così via. A questo punto possiamo riscrivere la Formula 3.1.3 in termini di prodotto scalare attraverso le Haar

wavelets di primo livello:

$$d_m = f \cdot W_m^1 \tag{3.1.9}$$

per $m = 1, 2, \ldots, N/2$.

Possiamo utilizzare l'idea del prodotto scalare per riaffermare la proprietà di Small Fluctuations in maniera più precisa. Chiamiamo *supporto* di una Haar wavelet l'insieme dei due indici dove la wavelet è diversa da zero, così facendo otteniamo una versione più precisa del principio di Small Fluctuation, in particolare:

Proprietà 1. *Se un segnale f ha (approssimativamente) un supporto costante per la Haar wavelet W_k^1, allora il valore rispettivo nel sottosegnale fluctuation $d_k = f \cdot W_k^1$ è (approssimativamente) zero.*

Possiamo esprimere i valori del sottosegnale *trend* di primo livello come prodotto scalare di certi segnali elementari. Questi segnali elementari vengono chiamati *Haar scaling signals* di primo livello e sono definiti come segue:

$$\begin{aligned} V_1^1 &= \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, \ldots, 0\right) \\ V_2^1 &= \left(0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, \ldots, 0\right) \\ &\vdots \\ V_{N/2}^1 &= \left(0, 0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \end{aligned}$$

Utilizzando questi *Haar scaling signals,* i valori $a_1, \ldots, a_{N/2}$ del primo sottosegnale *trend* possono essere rappresentati mediante prodotti scalari:

$$a_m = f \cdot V_m^1 \tag{3.1.10}$$

per $m = 1, 2, \ldots, N/2$.

Gli *Haar scaling signals* sono simili alle Haar wavelets. Essi hanno tutti un'energia uguale a 1 e un *supporto* consistente di soli due indici consecutivi.

In effetti, essi sono tutti traslazioni di un numero pari di unità del primo valore di *scaling signal* V_1^1. Diversamente dalle Haar wavelets, la media dei valori dei *Haar scaling signals* non è zero, ognuno di essi ha un valore medio di $1/\sqrt{2}$.

L'idea discussa sopra può essere estesa a tutti i livelli. Vediamo per semplicità solamente il secondo livello. Il secondo livello delle *Haar scaling signals* è definito come:

$$\begin{array}{rcl} V_1^2 & = & \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0, \ldots 0\right) \\ V_2^2 & = & \left(0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0, \ldots 0\right) \\ & \vdots & \\ V_{N/2}^2 & = & \left(0, 0, \ldots, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right). \end{array} \tag{3.1.11}$$

Questi *scaling signals* sono tutti traslazioni di quattro unità del primo valore V_1^2, hanno tutti energia uguale ad 1 ed un valore medio di 1/2. Inoltre, i valori del sottosegnale *trend* di secondo livello a^2 sono prodotti scalari degli *scaling signals* con il segnale *f*. Quindi a^2 soddisfa la seguente formula:

$$a^2 = (f \cdot V_1^2, f \cdot V_2^2, \ldots, f \cdot V_{N/4}^2). \tag{3.1.12}$$

Allo stesso modo, le Haar wavelets di secondo livello sono definite come:

$$\begin{array}{rcl} W_1^2 & = & \left(\frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}, 0, 0, \ldots, 0\right) \\ W_2^2 & = & \left(0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}, 0, 0, \ldots, 0\right) \\ & \vdots & \\ W_{N/4}^2 & = & \left(0, 0, \ldots, 0, \frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}\right) \end{array} \tag{3.1.13}$$

Queste wavelts hanno tutte *supporto* di lunghezza 4. Esse hanno tutte un'energia pari a 1 ed un valore medio 0. Utilizzando il prodotto scalare, il sottosegnale *fluctuation* di secondo livello d^2 soddisfa la seguente formula:

$$d^2 = (f \cdot W_1^2, f \cdot W_2^2, \ldots, f \cdot W_{N/4}^2). \tag{3.1.14}$$

3.2 Wavelet: applicazioni

Le wavelet sono uno strumento matematico per la decomposizione gerarchica di funzioni, indipendentemente dal fatto che la funzione di interesse sia un'immagine, una curva, una superficie, etc. esse offrono una tecnica elegante per rappresentare i diversi livelli di dettaglio della funzione in maniera efficiente.

Negli ultimi anni abbiamo assistito ad un notevole aumento dell'utilizzo delle wavelet in diverse aree dell'informatica.

Il campo di utilizzo più evidente è la compressione dei segnali digitali, anche se vengono ampiamente sfruttate nel campo delle immagini digitali. Sino al 2000 lo standard JPEG si basava sulla trasformata coseno, ora il nuovo standard impiega le trasformate wavelet.

Tuttavia in questa tesi, ci concentreremo sulla applicazione delle wavelet per nella rappresentazione dei dati e per le range query.

3.2.1 Wavelet e range query

Le wavelets vengono utilizzate nei database [23][24][25] per la rappresentazione di informazioni registrate nel database attraverso istogrammi. Gli istogrammi sono utilizzati per sintetizzare la distribuzione dei dati presenti all'interno del database permettendo così al *query optimizers* di stimare "l'ampiezza di una query" intesa come la stima del numero di record presenti nel database che soddisfano la query. Evidentemente migliore è la qualità di un istogramma migliori saranno i risultati restituiti. Grazie alla decomposizione wavelets è possibile costruire istogrammi basati sulla distribuzione dei dati con una buona accuratezza nella stima con un uso limitato di memoria, tali istogrammi permettono di stimare range query in maniera efficiente ed efficace.

Vediamo in questa sezione come costruire un istogramma *gerarchico* e *globale* per range query attraverso le wavelets.

Notazioni

L'insieme dei predicati che utilizziamo per stimare la distribuzione dei dati è l'insieme dei *range predicates* della forma $l \leq X \leq h$ dove, X è un attributo non negativo del dominio della relazione R ed l e h sono costanti. Il predicato è soddisfatto da tutte quelle tuple il cui valore dell'attributo X è compreso tra l ed h (estremi compresi). Indichiamo inoltre con t una generica tupla della relazione e con $t.x$ il valore dell'attributo X della tupla t.

Adottiamo la seguente notazione per descrivere la distribuzione dei dati ed i vari istogrammi.

- Il *dominio* $D = \{0, 1, 2, ..., N-1\}$ di un attributo X è l'insieme di tutti i possibili valori di X.
- L'insieme dei valori $V \subseteq D$ è costituito dagli n valori distinti di X che sono presenti nella relazione R.

Poniamo $v_1 < v_2 < \ldots < v_n$ come n valori di V.

- Lo *spread* s_i di v_i è definito come $s_i = v_{i+1} - v_i$ (dove $s_0 = v_1$).
- La *frequenza* f_i di v_i è il numero di tuple per cui X ha valore v_i.
- La *frequenza cumulata* c_i di v_i è il numero di tuple $t \in R$ tali per cui $t.X \leq v_i$; in altri termini $c_i = \sum_{j=1}^{i} f_j$.
- La distribuzione dei dati di X è l'insieme delle coppie:

$$T^C = \{(v_1, c_1), (v_2, c_2), \ldots, (v_n, c_n)\}$$

- La distribuzione dei dati estesa di X, indicata con T^{C+}, è la distribuzione dei dati T^Cestesa sull'intero dominio D assegnando frequenza 0 a tutti i valori compresi nell'insieme $D - V$.

Costruire un istogramma: descrizione ad alto livello

Ad alto livello la costruzione di un istogramma basato sulle wavelet è un processo che si compone di tre fasi principali:

1. *Preprocessing:* nella fase di preprocessing viene calcolata la distribuzione dei dati estesa T^{C+} dell'attributo X sulla base dei dati originali.

2. *Wavelet Decomposition*: una volta ottenuta la distribuzione dei dati estesa calcoliamo la wavelet decomposition di T^{C+}ottenendo un insieme di N coefficienti wavelet.

3. *Pruning: n*ella terza ed ultima fase manteniamo solamente gli m coefficienti wavelet più significativi per un qualsiasi valore di m che rappresenti la quantità di memoria dedicata alla computazione delle wavelet. La scelta di quali coefficienti mantenere dipende dal particolare algoritmo di *pruning* adottato.

Dopo aver applicato l'algoritmo sopra descritto otteniamo m coefficienti wavelet. I valori e gli indici degli m coefficienti selezionati vengono memorizzati in una struttura dati H così da formare un istogramma. Tale istogramma viene utilizzato successivamente nella fase di query per la ricostruzione della distribuzione dei dati approssimata.

A questo punto per calcolare la stima del numero di tuple il cui valore X appartiene all'intervallo $l \leq X \leq h$ procediamo come segue: utilizzando gli m coefficienti ricostruiamo il valore approssimato che h e $l-1$ hanno nella funzione di distribuzione dei dati estesa, grazie a questi valori eseguendo la sottrazione del valore ottenuto per $l-1$ a quello per h otteniamo una approssimazione del numero di elementi appartenenti all'intervallo $[l, h]$.

Preprocessing

Nella fase di preprocessing calcoliamo la distribuzione dei dati T complessiva. Il calcolo esatto di T può essere eseguito mantenendo un contatore per ogni

valore distinto in V. La distribuzione dei dati estesa T^{C+} può essere calcolata facilmente partendo da T e ponendo frequenza 0 ai valori non presenti in V.

Wavelet decomposition

Come risultato della fase di preprocessing otteniamo la distribuzione dei dati estesa T^{C+} che può essere rappresentata mediante un array S monodimensionale di grandezza N. Lo scopo dell'algoritmo di wavelet decomposition è quello di riuscire a rappresentare l'istogramma a diversi livelli gerarchici di dettaglio.

Per mostrare il funzionamento dell'algoritmo di wavelet decomposition, implementando la funzione Haar wavelet, utilizziamo un semplice esempio.

Supponiamo che un attributo X assuma i seguenti valori:

$$\{0, 0, 2, 3, 3, 4, 4, 5, 5, 6, 7, 7\}$$

Ricaviamo le frequenze per ogni valore dell'attributo X:

$$\{(0, 2), (2, 1), (3, 2), (4, 2), (5, 2), (6, 1), (7, 2)\}$$

A questo punto otteniamo la distribuzione dei dati T^C:

$$T^C = \{(0, 2), (2, 3), (3, 5), (4, 7), (5, 9), (6, 10), (7, 12)\}$$

Come descritto sopra possiamo ricavare la distribuzione dei dati estesa ponendo frequenza 0 per i valori del dominio non assunti da alcuna istanza dell'attributo X ottenendo così:

$$T^{C+} = \{(0, 2), (1, 2), (2, 3), (3, 5), (4, 7), (5, 9), (6, 10), (7, 12)\}.$$

La distribuzione dei dati estesa appena ricavata può essere rappresentata mediante un array S di $N = 8$ elementi:

$$S = [2, 2, 3, 5, 7, 9, 10, 12] .$$

A questo punto possiamo procedere con la decomposizione wavelet; per prima cosa calcoliamo la media delle frequenze cumulate coppia a coppia, in tal modo otteniamo un nuovo array (formato solamente dalle medie delle frequenze) di $N = 4$ elementi:

$$[2, 4, 8, 11] .$$

Ovviamente tramite questo procedimento alcune informazioni vengono perse. Per riuscire a ricostruire il segnale originale, partendo dai valori ottenuti dal calcolo dalle medie, abbiamo la necessità di mantenere una struttura dati che catturi le informazioni mancanti. Le Haar wavelets memorizzano le differenze tra le coppie dei valori originali (dividendole per 2) come *detail coefficients.* Nell'esempio precedente i quattro *detail coefficients* sono: $(2-2)/2 = 0$, $(3-5)/2 = -1$, $(7-9)/2 = -1$, $(10-12)/2 = -1$.

Nell'esempio precedente siamo riusciti a decomporre il segnale originale in un segnale di minor risoluzione, in particolare abbiamo ottenuto metà elementi con i relativi detail coefficients. Ripetendo questo procedimento ricorsivamente sulle medie calcolate, otteniamo una decomposizione completa:

Risoluzione	Media	Detail Coefficients
8	$[2, 2, 3, 5, 7, 9, 10, 12]$	
4	$[2, 4, 8, 11]$	$[0, -1, -1, -1]$
2	$\left[3, 9\frac{1}{2}\right]$	$\left[-1, -1\frac{1}{2}\right]$
1	$\left[6\frac{1}{4}\right]$	$\left[-3\frac{1}{4}\right]$

Definiamo decomposizione wavelet degli otto valori appartenenti all'attributo X come l'elemento rappresentante la media complessiva di tutti i valori accompagnato dai details coefficients in ordine crescente di risoluzione. Riferendoci all'esempio precedente, la decomposizione wavelet del nostro segnale

originale è data da:

$$\hat{S} = \left[6\tfrac{1}{4}, -3\tfrac{1}{4}, -1, -1\tfrac{1}{2}, 0, -1, -1, -1\right].$$

Rispetto all'array rappresentante la distribuzione dei dati estesa nessuna informazione è stata aggiunta o persa, l'array originale aveva otto elementi così come l'array ottenuto dopo il processo di wavelet decomposition.

La decomposizione wavelet è computazionalmente molto efficiente dato che richiede solo $O(N)$ per elaborare un segnale di dimensione N.

Pruning dei coefficienti

A seconda della quantità di memoria dedicata alla computazione delle wavelet o per motivi di ottimizzazione abbiamo la necessità di mantenere solamente un certo numero degli N coefficienti ottenuti. Poniamo m come il numero massimo di coefficienti wavelet che possiamo mantenere; il resto dei coefficienti saranno eliminati dalla trasformazione wavelet. Tipicamente avremo $m \ll N$. Lo scopo della fase di pruning è quello di determinare quali sono i "migliori" m coefficienti da mantenere in modo da minimizzare l'errore nell'approssimazione.

Il problema è definire quali sono i "migliori" m coefficienti eliminando gli $N - m$ coefficienti meno significativi. A questo scopo è necessario definire cosa significa "migliori" e che tipo di misurazione dell'errore utilizzeremo così che la fase di pruning minimizzi l'errore di approssimazione.

Possiamo misurare l'errore di approssimazione in diversi modi.

Sia v_i la risposta reale ad un query q_i e $\hat{v_i}$ la risposta approssimata. Ad esempio supponiamo di avere $10 \leq x \leq 30$ con 5 valori che soddisfano la query. Occorre verificare quanti valori vengono restituiti utilizzando l'approssimazione, considerando che ci può essere una sottostima o una sovrastima.

La seguente tabella mostra alcune metriche per la misurazione dell'errore:

	Simbolo	Definizione
absolute error	e_i^{abs}	$\lvert v_i - \hat{v_i} \rvert$
relative error	e_i^{rel}	$\frac{\lvert v_i - \hat{v_i} \rvert}{\max\{1, v_i\}}$
modified relative error	$e_i^{m\_rel}$	$\frac{\lvert v_i - \hat{v_i} \rvert}{\max\{1, \min\{v_i, \hat{v_i}\}\}}$
combined error	e_i^{comb}	$\min\left\{\alpha \times e_i^{abs}, \beta \times e_i^{rel}\right\}$
modified combined error	$e_i^{m\_comb}$	$\min\left\{\alpha \times e_i^{abs}, \beta \times e_i^{m\_rel}\right\}$

I parametri α e β sono positivi e costanti.

In questo caso la definizione di *relative error* è leggermente diversa da quella tradizionale la quale non è definita quando $v_i = 0$. Infatti se $v_i = 0$, quindi non ci sono risultati per la query, l'errore relativo è comunque pari al numero di elementi nella approssimazione. Consideriamo il *modified relative error*. Poichè al numeratore c'è la differenza tra la risposta alla query e quella approssimata, ma al denominatore viene posto il minimo tra la risposta della query e quella approssimata, è evidente che questa metrica "premia" le sovrastime, piuttosto che le sottostime, come vedremo nell'esempio seguente.

Esempio. Supponiamo che la dimensione esatta di una query sia $v_i = 10$. La dimensione dopo il pruning sia $\hat{v_i} = 5$ o $\hat{v_i} = 20$ ognuna delle approssimazioni ha lo stesso *modified relative error,* cioè $5/5 = 10/10 = 1$, anche se la differenza tra la dimensione esatta e quella approssimata è maggiore nel secondo caso. Invece, in termini di errore relativo, la stima $\hat{v_i} = 5$ ha il *relative error* di $5/10 = 0.5$, e la stima $\hat{v_i} = 20$ ha il *relative error* di $10/10 = 1$. La stima $\hat{v_i} = 0$ ha il *relative error* di $10/10 = 1$, mentre il *modified relative error* è di $10/1 = 10$. Il *combined error* riflette l'importanza di avere sia un buon *relative error* che un buon *absolute error* per ogni approssimazione.

Una volta scelta quale delle misure sopra illustrate rappresenti l'errore della query individuale dobbiamo scegliere una norma che misuri l'errore di un insieme di queries. Rappresentiamo con $e = (e_1, e_2, \ldots, e_Q)$ il vettore degli errori ricavati eseguendo una serie Q di queries. Assumiamo che una volta scelta la misura dell'errore questa sia utilizzata per tutte le queries appartenenti a Q.

Ad esempio, per l'*absolute error*, possiamo scrivere $e_i = e_i^{abs}$.

Definiamo l'errore globale per le Q queries con una delle seguenti misure:

	Simbolo	Definizione
norma 1	$\|e\|_1$	$\frac{1}{Q}\sum_{1\leq i\leq Q} e_i$
norma 2	$\|e\|_2$	$\sqrt{\frac{1}{Q}\sum_{1\leq i\leq Q} e_i^2}$
norma infinito	$\|e\|_\infty$	$\max_{1\leq i\leq Q}\{e_i\}$

Il primo passo da fare nella fase di pruning è quello di attribuire un peso ai coefficienti. In particolare possiamo pesare i coefficienti attraverso la normalizzazione. Per le Haar wavelet la normalizzazione è ottenuta dividendo i coefficienti $\hat{S}(2^j), \ldots, \hat{S}(2^{j+1}-1)$ per $\sqrt{2^j}$ per ogni $0 \leq j \leq \log N - 1$.

Una volta pesati i coefficienti possiamo eseguire uno dei seguenti metodi di pruning:

1. Scegliere gli m coefficienti più grandi in valore assoluto.
2. Scegliere gli m coefficienti in modo *greedy*. Ad esempio, potremmo scegliere gli m coefficienti più grandi in valore assoluto e successivamente ripetere i due passaggi seguenti m volte:
   (a) scegliere il coefficiente la cui inclusione comporta la più grande riduzione dell'errore;
   (b) scartare i coefficienti la cui eliminazione comporta al più piccolo incremento dell'errore.
3. Due varianti del metodo greedy possono essere:
   (a) iniziare con *gli m/2* coefficienti più grandi in valore assoluto e scegliere i seguenti *m/2* in modo *greedy*;
   (b) iniziare con *2m* coefficienti più grandi in valore assoluto ed eliminarne m in maniera *greedy*.

E' dimostrato che se le funzioni base wavelet sono ortonormali allora il metodo 1 è ottimale per la misura media dell'errore con *norma 2*. Tuttavia, per tutte le wavelet non ortogonali e per altre norme non è stata ancora trovata una tecnica efficace per scegliere i migliori m coefficienti.

Alla fine della fase di pruning otteniamo m coefficienti wavelet significativi. Questi coefficienti, insieme ai loro indici, formano l'istogramma wavelet. Indichiamo con H l'istogramma calcolato da un array monodimensionale S di lunghezza N. Possiamo considerare H come un array monodimensionale di lunghezza m, in cui ogni entry ha un coefficiente wavelet v_j ed un indice i_j.

$$H[j] = (i_j, v_j), \qquad 1 \leq j \leq m.$$

Range query su istogrammi wavelet

Nella fase di query è necessario stimare il risultato di una range query del tipo $l \leq X \leq h$. La strategia utilizzata è quella di ricostruire la frequenza cumulata di $l-1$ ed h, indicate rispettivamente con $S(l-1)$ e $S(h)$, utilizzando gli m coefficienti ottenuti grazie all'algoritmo di wavelet decomposition. Il risultato della query viene così stimato calcolando $S(h) - S(l-1)$.

Come vedremo nel capitolo 4, in un sistema P2P come HASP, è importante minimizzare la banda utilizzata per le trasmissioni dei dati tra nodi. Per questa ragione ogni nodo calcolerà l'istogramma contenente di detail coefficients e spedirà questo digest agli altri nodi. Per calcolare la range query, ogni nodo dovrò tuttavia ricostruire le frequenze cumulate a partire dall'istogramma. In questo paragrafo esaminiamo quindi il problema della ricostruzione delle frequenze cumulate a partire dall'istogramma.

Per comprendere meglio l'algoritmo utilizzato, esaminiamo la relazione tra i coefficienti della wavelet e la ricostruzione del dato originale utilizzando una struttura gerarchica, l'*error tree*. Tale struttura è costruita in base alla procedura di wavelt decomposition. In Figura 3.3 è mostrato un esempio di *error tree* basato sull'applicazione delle wavelet decomposition all'array di frequenze cumulate $S = [2, 2, 3, 5, 7, 9, 10, 12]$. Ad ogni nodo interno dell'al-

bero è associato un *detail coefficient*, mentre alle foglie vengono associati i valori delle frequenze cumulate calcolati di volta in volta.

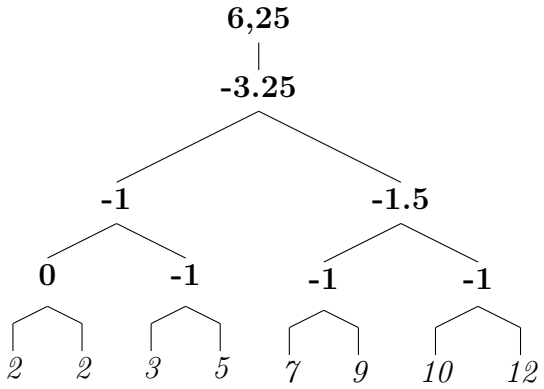


Figura 3.3: Error tree

Il procedimento per la creazione dell'*error tree* è un procedimento bottom-up e rispecchia esattamente il processo di wavelet decomposition. In particolare viene costruito e rappresenta in modo gerarchico l'istogramma H risultato della wavelet decomposition.

Vediamo come, con l'ausilio della struttura *error* tree, vengono ricostruiti i valori originali partendo dai coefficienti.

Utilizziamo $\widehat{S}(i)$ per indicare il valore di un nodo interno i e $S(j)$ per rappresentare il valore della foglia j, con $left(i)$ *e* $right(i)$ indichiamo il figlio sinistro e destro di un nodo i mentre con $leaves(i)$ l'insieme di nodi appartenenti al sottoalbero con radice i. Il valore medio dei nodi appartenenti a $leaves(i)$ è rappresentato da $ave\_leaf\_val(i)$. Per qualsiasi foglia i, utilizziamo $path(i)$ per indicare l'insieme dei nodi interni appartenenti al percorso da i alla radice.

Di seguito sono riportate alcune proprietà della struttura *error tree* sfruttate nell'algoritmo:

Lemma 1 *Per qualsiasi nodo interno, non radice, abbiamo*

$$\hat{S}(i) = \frac{ave\_leaf\_val(left(i)) - ave\_leaf\_val(right(i))}{2}.$$

Eempio. Consideriamo il nodo corrispondente al coefficiente -3,25, sull'albero dei coefficienti.

$$ave\_leaf\_val(left(i)) = \frac{2+2+3+5}{4} = \frac{12}{4} = 3$$

$$ave\_leaf\_val(right(i)) = \frac{7+9+10+12}{4} = -\frac{38}{4} = 9,5$$

$$\frac{3-9,5}{2} = -,325$$

Questa proprietà permette di ricostruire un qualsiasi coefficiente a partire dalla distribuzione dei dati in input.

Lemma 2 *La ricostruzione di una qualsiasi foglia $S(i)$ dipende solamente dai nodi presenti in $path(i)$, cioè presenti sul cammino dalla radice a quella foglia.*

Lemma 3 *Qualsiasi dato originale $S(i)$ può essere espresso in termini di tutti i coefficienti appartenenti a $path(i)$ nel seguente modo*:

$$S(i) = \sum_{j \in path(i)} sign(i,j) * \hat{S}(j),$$

dove

$$sign(i,j) = \begin{cases} 1 & se\ j = 0\ oppure\ se\ la\ foglia\ i\ appartiene\ al\ sottoalbero\ sinistro\ di\ j \\ -1 & se\ la\ foglia\ i\ appartiene\ al\ sottoalbero\ destro\ di\ j \end{cases}$$

La somma descritta sopra avviene su tutti i nodi x appartenenti a $path(i)$.

Consideriamo la ricostruzione di una qualsiasi foglia $S(i)$. Tale procedimento è la somma algebrica di tutti i nodi interni appartenenti a $path(i)$. Ad esempio, per $i = 1$ abbiamo:

$$S(1) = \hat{S}(0) + \hat{S}(1) + \hat{S}(2) - \hat{S}(4).$$

In generale, qualsiasi valore $S(i)$ può essere rappresentato dalla somma algebrica dei coefficienti che si trovano lungo il percorso $path(i)$. Qualsiasi nodo interno non radice contribuisce positivamente alle foglie appartenenti al suo sottoalbero sinistro e negativamente a quelle appartenenti al sottoalbero destro.

Esempio. Consideriamo ancora la figura 3.3, in particolare la foglia 9:

$$9 = 6,25 + 3,25 - 1,25 + 1.$$

Si parte dalla radice con valore 6,25 e si segue il cammino fino alla foglia sommando algebricamente i coefficienti.

Il risultato riportato nel Lemma 3 ci consente di ricostruire ogni elemento della distribuzione cumulata a partire dall'*error tree*. Vediamo un'applicazione.

Esempio. Consideriamo la decomposizione wavelet descritta nella sezione precedente, per cui supponiamo di avere un attributo X definito nel dominio $[0; 8]$ con i seguenti valori:

$$\{0, 0, 2, 3, 3, 4, 4, 5, 5, 6, 7, 7\}$$

applicando la procedura descritta sopra otteniamo una decomposizone wavelet uguale a:

$$\hat{S} = \left[6\tfrac{1}{4}, -3\tfrac{1}{4}, -1, -1\tfrac{1}{2}, 0, -1, -1, -1\right]$$

Assumiamo di non effettuare nessuna tecnica di pruning; a questo punto per poter eseguire range query sull'attributo X rappresentiamo la decomposizione wavelet attraverso la struttura *error tree* ottenendo:

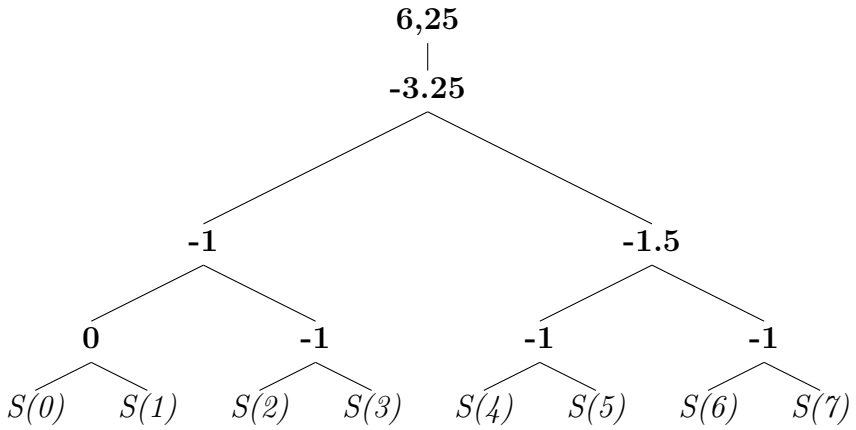


Figura 3.4: Error tree

Sfruttiamo le proprietà dell'albero *error tree* per eseguire range query sull'attributo X. Per stimare la range query $3 \leq x \leq 7$ sull'attributo X ricostruiamo il valore delle foglie $S(2)$ e $S(7)$ nella struttura *error tree* ottenendo:

$$S(2) = \hat{S}(0) + \hat{S}(1) - \hat{S}(2) + \hat{S}(5) = 6,25 - 3,25 + 1 - 1 = 3$$
$$S(7) = \hat{S}(0) - \hat{S}(1) - \hat{S}(3) - \hat{S}(7) = 6,25 + 3,25 + 1,5 + 1 = 12$$

A questo punto per stimare la query $3 \leq x \leq 7$ eseguiamo il calcolo:

$$S(7) - S(2) = 12 - 3 = 9$$

Abbiamo stimato il numero di valori che cadono nell'intervallo individuato dalla range query.

3.3 Bloom Filters

3.3.1 Descrizione

Il Bloom Filter [27] è una struttura dati compatta, ideata da Burton Bloom nel 1970, utilizzata per la rappresentazione probabilistica di un insieme al

fine di supportare queries di appartenenza. I Bloom Filters, a causa della loro rappresentazione compatta, possono restituire falsi positivi, cioè alcune queries potrebbero riconoscere in modo non corretto che un elemento appartiene ad un insieme; ma quando la probabilità di errore è mantenuta sotto una certa soglia permettono di ottenere dei trade-off interessanti tra lo spazio impiegato in memoria per la rappresentazione dell'insieme e la probabilità di falsi positivi. Essi vengono utilizzati nel campo dei database e negli ultimi anni sono diventati popolari anche in altri rami dell'informatica in particolare nel networking.

3.3.2 Concetti matematici

Un Bloom Filter rappresenta un insieme $S = (x_1, x_2, \ldots, x_n)$ di n elementi attraverso un array di m (con $m \ll n$) bits inizialmente posti tutti a 0. Esso utilizza k funzioni hash indipendenti $h_1, \ldots, h_k$ che producono un valore distribuito uniformemente nel range $[1, \ldots, m]$. Per ogni elemento $x \in S$, i bits nella posizione $h_i(x)$ vengono impostati a 1 per $1 \leq i \leq k$. Un bit può essere settato a 1 più di una volta. Per verificare se un dato elemento y appartiene all'insieme S mappato sul Bloom Filters, applichiamo le k funzioni hash ad y. Se almeno una posizione restituita da una funzione hash $h_i(y)$ è pari a 0 allora sicuramente l'elemento non è presente nell'insieme *S;* altrimenti se tutte le $h_i(y)$ posizioni sono impostate a 1, assumiamo che l'elemento appartenga all'insieme con una certa probabilità di errore. Un Bloom Filter può quindi produrre un *falso positivo,* le diverse funzioni hash calcolate su valori diversi possono portare allo stesso risultato; in particolare nella fase di look up di un elemento non appartenente all'insieme può accadere che una funzione hash restituisca l'indice di una posizione impostata ad 1 da un altro elemento.

Ad esempio, supponiamo di avere un insieme S di 2 elementi $S = \{x_1, x_2\}$ con $k = 3$ funzioni hash ed un Bit-Vector B di lunghezza $m = 10$. Inizialmente tutti i bit del Bit-Vector B saranno impostati a 0. A questo punto, ad ogni elemento appartenente all'insieme S verranno applicate le k funzioni hash,

ogni funzione hash restituirà una posizione del Bit-Vector B la quale sarà impostata a 1. In Figura 3.5 è mostrata l'operazione appena descritta.

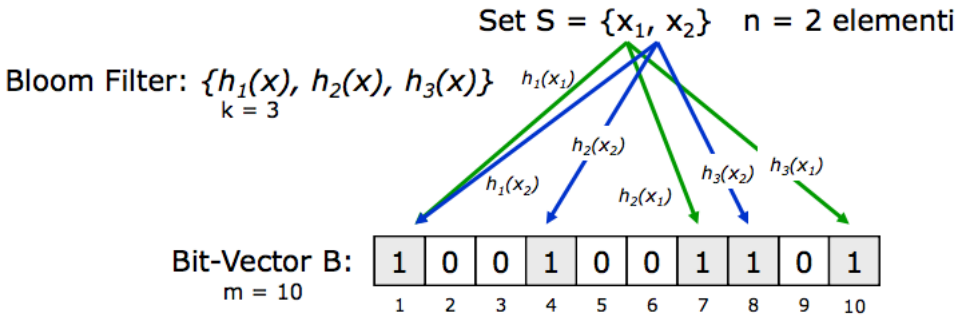


Figura 3.5: Bloom filter operazione di add

Per verificare che un generico elemento y appartenga all'insieme S, le k funzioni hash vengono applicate ad y.

$$z_1 = h_1(y), z_2 = h_2(y), z_3 = h_3(y)$$

L'elemento y appartiene ad S se e solo se tutti gli elementi $B[z_1], B[z_2], B[z_3]$ del Bit-Vector B sono posti a 1.

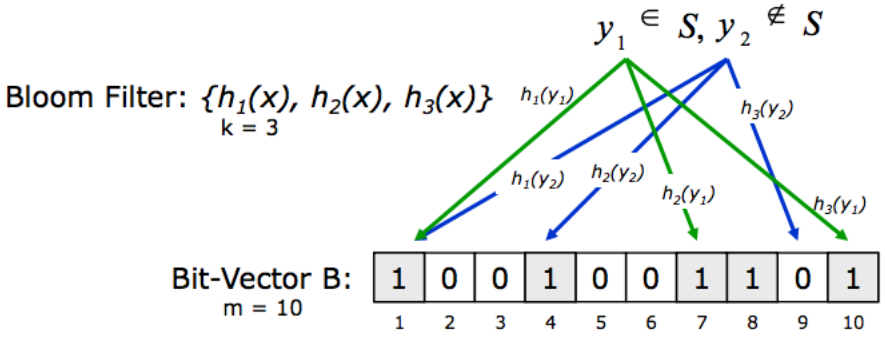


Figura 3.6: Bloom Filter operazione di look up

Dati due Bloom Filters *B1* e *B2* che rappresentano, rispettivamente gli insiemi *S1* e *S2* mediante lo stesso numero di bits e lo stesso numero di funzioni hash, è possibile ottenere:

- il Bloom Filter che rappresenta l'*unione* dei due insiemi $(S1 \cup S2)$ calcolando l'OR bit a bit dei due Bloom Filters *B1* e *B2;*
- il Bloom Filter che rappresenta l'*intersezione approssimata* dei due insiemi $(S1 \cap S2)$ eseguendo l'AND bit a bit dei due Bloom Filters *B1* e *B2;* l'intersezione ottenuta è un'approssimazione perché un bit può assumere valore uguale a 1 in entrambi i Bloom Filters nel caso in cui:
  - quel bit corrisponde ad un elemento $\in S1 \cap S2$,
  - quel bit corrisponde sia ad un elemento $\in S1 - (S1 \cap S2)$ che ad un elemento $\in S2 - (S1 \cap S2)$ e quindi non corrisponde ad un elemento dell'intersezione.

L'operazione di inserimento di un elemento nel Bloom Filter non è invertibile, non è corretto azzerare tutti i bits corrispondenti all'applicazione delle k funzioni hash all'elemento che si vuol eliminare. I *Counting Bloom Filters* consentono di superare tale limitazione sostituendo i bit del Bit-Vector con dei contatori (solitamente di 3-4 bits). A questo punto quando viene inserito un nuovo elemento nel Counting Bloom Filter i k contatori vengono incrementati di 1, allo stesso modo quando un elemento viene rimosso, i rispettivi k contatori vengono decrementati di 1.

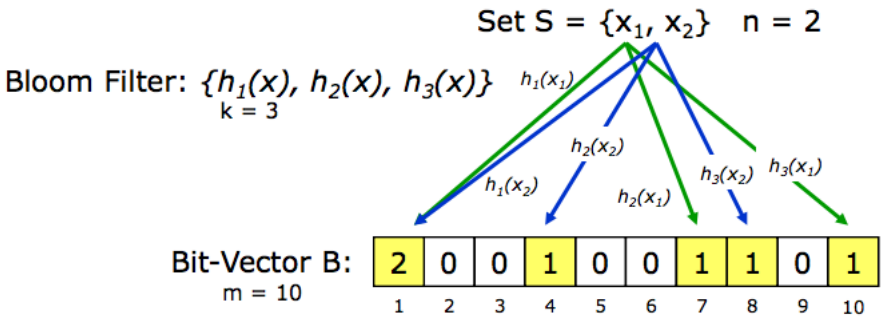


Figura 3.7: Counting Bloom Filter operazione di add

3.3.3 Probabilità di falsi positivi

Facendo l'assunzione che le funzioni hash siano perfettamente random possiamo stimare la probabilità di ottenere un falso positivo per un elemento non presente nell'insieme S.

Dopo che a tutti gli elementi di S è stata applicata la funzione hash e quindi sono stati inseriti tutti nel Bloom Filter, la probabilità che uno specifico bit sia ancora 0 è:

$$p' = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-km/n} \tag{3.3.1}$$

Assumiamo adesso di aver costruito un Bloom Filter; sappiamo che la percentuale di bit ancora a 0 è $e^{-km/n}$, applicando le k funzioni hash ottengo un falso positivo se per k volte individuo un bit a 1:

$$f = (1 - e^{-km/n})^k = (1 - p')^k = \epsilon \tag{3.3.2}$$

Fissati n ed m, al crescere di k la probabilità p' che un bit sia 0 diminuisce, rendendo più probabile la collisione di una singola funzione hash; ma allo stesso tempo con k aumenta il numero di collisioni necessarie per avere un falso positivo. Per trovare il minimo poniamo $g = \ln f = k \ln(1 - e^{-km/n})$ trovando il minimo in funzione di k derivando:

$$\begin{aligned}\frac{dg}{dk} &= \ln(1 - e^{-km/n}) + \frac{k}{(1 - e^{-km/n})}\left(-e^{-km/n}\right)\left(-\frac{m}{n}\right) \\ &= \ln(1 - e^{-km/n)} + \frac{km}{n}\frac{e^{-km/n}}{1 - e^{-km/n}}\end{aligned}$$

e ponendo la derivata a 0:

$$\begin{aligned}\frac{dg}{dk} = 0 &\Longrightarrow (1 - e^{-km/n}) \ln(1 - e^{-km/n}) = -\frac{km}{n} e^{-km/n} \\ &\Longrightarrow 1 - e^{-km/n} = e^{-km/n} \\ &\Longrightarrow k = (\ln 2)\frac{n}{m}\end{aligned}$$

Questo è il punto di minimo globale (trascurando il fatto che k deve essere intero), corrisponde ad una probabilità di falsi positivi di:

$$f = \left(\frac{1}{2}\right)^k = \left(\frac{1}{2}\right)^{(\ln 2)n/m} \approx (0.6185)^{n/m}.$$

Questa probabilità è esponenziale decrescente all'aumentare di m/n (numero di bit del Bloom Filter per ogni elemento) vedi Figura 3.8.

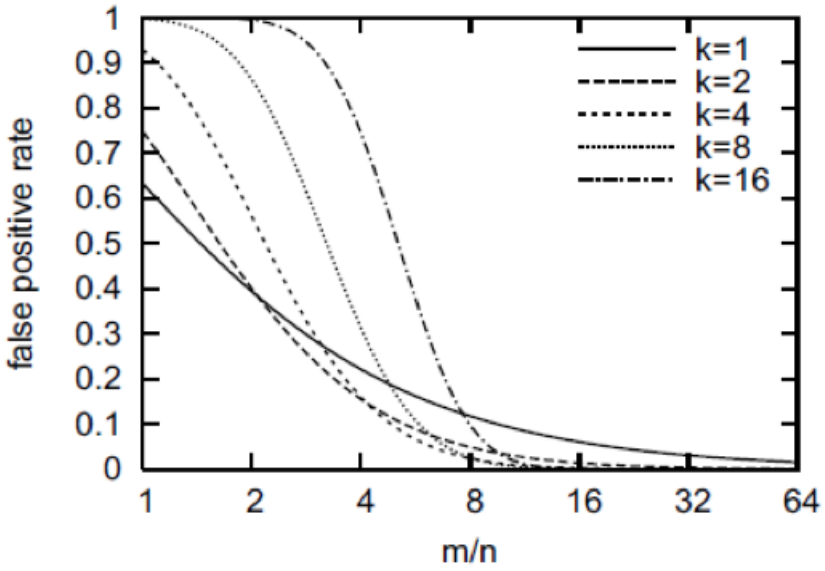


Figura 3.8: Probabilità falsi positivi all'aumentare dei bit assegnati ad ogni elemento

Se si desidera che questa probabilità sia inferiore ad una certa costante c, è sufficiente porre:

$$n \geq m \frac{\ln c}{|\ln 0.6185|}$$

In Figura 3.9 è possibile visualizzare l'andamento della probabilità al variare degli elementi dell'insieme, con k ottimo e m variabile al variare di n. Dal grafico possiamo notare che se il numero di bit del Bloom Filters non è sufficiente la probabilità di avere falsi positivi cresce esponenzialmente.

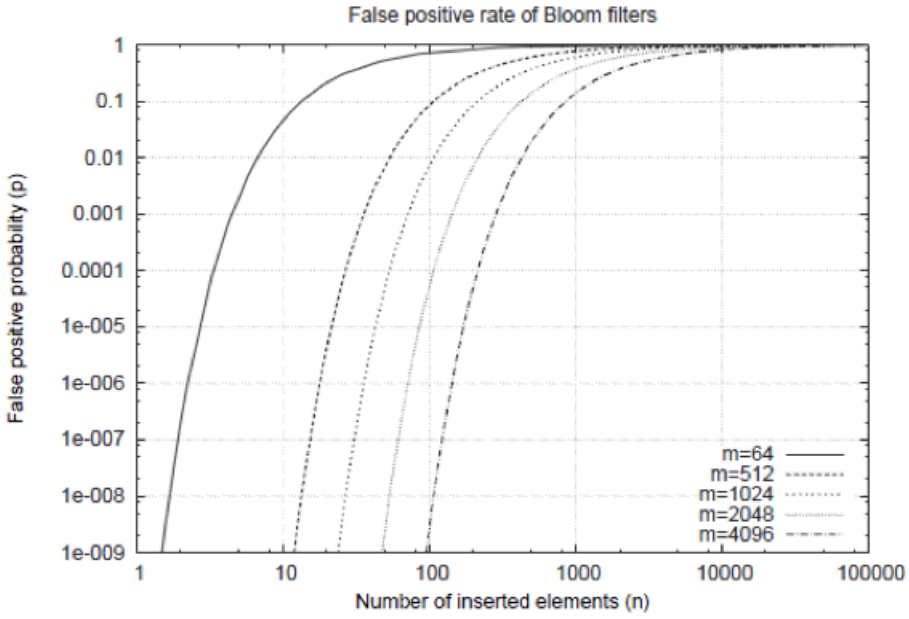


Figura 3.9: Probabilità falsi positivi al variare degli elementi dell'insieme

Con i valori ottimali calcolati in precedenza, la probabilità che uno dei bit sia rimasto a 0 dopo l'applicazione delle k funzioni è:

$$p = e^{-kn/m} = \frac{1}{2}$$

i valori ottimi si ottengono quando la probabilità che un bit sia rimasto a 0 dopo l'applicazione delle k funzioni hash agli elementi è $1/2$.

Un Bloom Filters ottimale è quindi riempito in modo tale da avere approssimativamente lo stesso numero di celle a 1 e a 0.

3.3.4 Utilizzo

L'efficienza sia in termini di spazio che di tempo di ricerca rendono il Bloom Filter un ottimo filtro per evitare accessi costosi a strutture di dati. Un esempio è il BigTable [28] di Google, un DBMS proprietario di Google, che impiega i Bloom Filter per evitare accessi al disco quando si richiedono righe

o colonne non esistenti. Anche le cache dei proxy Squid [29] sfruttano i Bloom Filters per poter stabilire rapidamente se un oggetto è presente nella cache del proxy tramite il suo URL.

I Bloom Filters possono inoltre essere utilizzati per verificare se una stringa fa parte di un particolare insieme. Per verificare l'esistenza di un match, si verifica se la stringa è presente nel Bloom Filter in cui sono stati inseriti gli elementi dell'insieme. Ad esempio, può essere utilizzato per verificare l'appartenenza di un URL ad una blacklist di siti malevoli, senza utilizzare l'intera blacklist. Se il riscontro è positivo, si puo procedere ad una ricerca vera e propria, ad esempio interrogando un server che ospiti la blacklist.

Infine, nella sincronizzazione di dati tra più entità, i Bloom Filter possono essere utilizzati per confrontare i dati posseduti da un'entità con quelli delle altre entità, e procedere all'invio dei dati mancanti.

Capitolo 4

Aggregazione di dati in HASP

In questo capitolo presentiamo le caratteristiche principali dell'architettura generale di *HASP*. *HASP* definisce un albero di aggregazione ed estende *XCone* al caso multidimensionale arricchendolo di strategie di aggregazione più sofisticate. Nella sezione 4.3 verranno mostrate le problematiche affrontate per l'integrazione di Wavelet e Bloom Filters in HASP.

4.1 XCone

XCone [31] è una rete P2P che nasce come estensione del sistema Cone descritto in precedenza il quale supporta solamente query del tipo *Trova k risorse maggiori di S*. In XCone diversamente, grazie ad una struttura ad albero distribuita, costruita al di sopra di una qualunque DHT, è possibile definire una strategia di aggregazione delle risorse tale per cui risulti efficiente risolvere query del tipo *Trova k risorse per cui il valore X sia compreso tra* $V \leq X \leq S$.

4.1.1 L'albero di mapping

In XCone l'aggregazione delle risorse è garantita dall'utilizzo di un albero binario, che offre la possibilità di sfruttare diverse strategie a seconda della

funzione di mapping adottata, basato sullo spazio degli identificatori della DHT.

Sono definiti *nodi logici* quei nodi dell'albero XCone che risultano essere mantenuti in modo distribuito tra i *nodi fisici* o *peer* della rete. I nodi logici foglia sono associati in modo univoco ai nodi fisici attraverso un processo di integrazione con la DHT che sarà illustrato in seguito, mentre, per quanto riguarda i nodi logici non foglia, l'associazione con i nodi fisici è effettuata per mezzo di una funzione di mapping. Il numero di nodi logici non foglia da associare a ciascun peer è determinato per mezzo di tale funzione di mapping e, nel caso pessimo, il numero di nodi logici assegnati ad un peer è inferiore ad un valore h pari al cammino dalla foglia alla radice dell'albero. La funzione di mapping ha un ruolo fondamentale nella costruzione dell'albero di XCone, in quanto a partire da due nodi logici a livello $l-1$ gestiti da due peer distinti, decide a quale dei due nodi assegnare la gestione del nodo logico di livello l.

In linea teorica, ad un qualsiasi peer può essere associato un qualsiasi nodo logico, ma per ottenere buone prestazioni in fase di entrata di un nodo nella rete (*join*) e di risoluzione delle query è necessario imporre alcuni vincoli. Ciascun peer è identificato da un ID univoco in formato binario e la sua posizione all'interno dell'albero è determinata in base al valore di tale identificatore. Ogni peer può gestire solo nodi logici che abbiano un ID tale da avere un prefisso comune ad esso. Questa regola di assegnazione dei nodi logici ai peer permette di implementare in modo efficiente la fase di *join*, in quanto il peer, che si unisce nell'albero XCone per la ricerca dei nodi logici da gestire, deve risalire l'albero a partire dalla foglia assegnata, evitando di visitare diversi sotto-alberi. Da ciò si deduce come l'assegnamento dei nodi foglia ai peer sia univoco, mentre, per quanto riguarda i nodi logici interni, i quali condividono il proprio prefisso con più nodi foglia, essi possono essere assegnati ad uno qualunque dei peer che corrispondono a tali foglie. Un ulteriore vincolo è dato dal fatto che il generico nodo P gestisca un insieme di nodi logici, che corrispondono ad un suffisso del proprio identificatore (ID). Il nodo P gestisce dunque una sequenza continua di nodi logici a partire dalla foglia da lui gestita, fino ad un certo livello l dell'albero in modo da ridurre

al minimo il numero di hops tra i diversi peer in fase di ricerca durante la risalita dell'albero XCone.

In XCone è definita un'altra funzione: *la funzione di digest*. Tale funzione associa a ciascun nodo una struttura in grado di sintetizzare le chiavi contenute nel sotto-albero radicato in esso. Esistono diversi tipi di funzioni di mapping in grado ciascuna di valutare aspetti diversi, quali il carico dei nodi logici gestiti da ogni nodo o la tipologia di banda. Ad esempio è possibile adottare una tecnica per migliorare il bilanciamento del carico in modo da regolare il numero di entrate significative presenti nella *routing table* di ciascun nodo. Nel momento in cui un nuovo nodo P entra in XCone, controlla se nel cammino che lega la foglia ad esso associata e la radice esistono nodi logici mappati a peer carichi. In tal caso il nodo P prende in gestione alcune entrate della tabella di routing di tali nodi. Questa operazione è naturalmente seguita da un aggiornamento del mapping di alcuni logici ai peer presenti sul cammino di P dalla foglia alla radice.

Nel caso di XCone la funzione di mapping adottata coincide con quella di Cone basata sulla scelta del peer avente in gestione la chiave di valore massimo ma, come vedremo in seguito, XCone definisce funzioni di digest diverse rispetto a Cone per supportare più efficacemente le range query.

L'assegnamento tra i peer della rete e i nodi foglia XCone viene realizzato a seguito dell'integrazione con la rete DHT sottostante. Se consideriamo una rete DHT con spazio degli identificatori (ID) di m bit, i nodi logici foglia sono assegnati ai peer attraverso un *trie*.

A ciascun peer in fase di *join* viene associato un identificatore di m bit in maniera casuale e ad ogni peer può essere associato un unico nodo foglia all'interno del *trie* in quanto l'albero XCone definisce un *trie* basato sui prefissi degli identificatori.

La Figura 4.1 illustra, attraverso un esempio, il mapping tra nodi logici foglia dell'albero XCone ed i peer di una rete DHT Chord con identificativi di 4 bits.

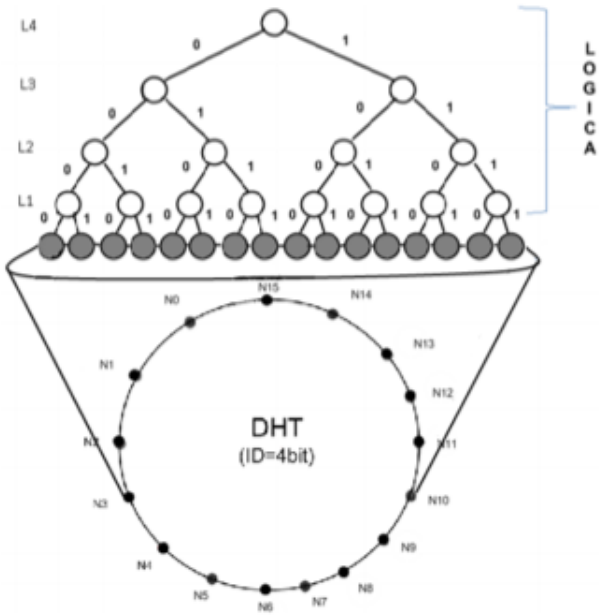


Figura 4.1: Overlay XCone-DHT

L'associazione nodo/foglia rimane fissata per tutta la permanenza del peer in rete. L'identificatore di ogni nodo è generato tramite una funzione hash impiegata dalla DHT per assegnare i peer alla rete e ciascun nodo gestisce in maniera indipendente la propria chiave, la quale può variare e non comporta uno spostamento del nodo all'interno dell'albero. Il dominio delle chiavi e quello degli identificatori risulta quindi essere distinto.

4.1.2 La tabella di routing

La tabella di routing è la struttura che mantiene l'albero di mapping distribuito tra i peer della rete. In tale struttura vengono memorizzati, per ciascun nodo, una lista di nodi da esso gestita. La routing table è composta al massimo da m entrate, una per ogni livello dell'albero, a partire dal livello 0.

La generica entrata E_l memorizza il risultato dell'applicazione della funzione di mapping f tra due nodi logici adiacenti di livello $l-1$. A tale posizione

corrisponde una coppia di valori (IP_1, IP_2), avente il seguente significato:

- IP_1: indirizzo del peer che gestisce il nodo logico di livello l;
- IP_2: indirizzo del peer che gestisce il nodo logico assegnato come figlio di livello $l-1$.

La mancanza di un limite inferiore alle chiavi contenute nel sotto-albero o più in generale la mancanza della conoscenza della distribuzione delle chiavi, potrebbe portare ad una visita infruttuosa o ad una perdita di nodi target nel caso di esplorazione o meno del sotto-albero. Per questo è stato necessario ampliare le informazioni rispetto a Cone, aggregando nel migliore dei modi le informazioni sulle chiavi presenti nei sotto-alberi. La generica entrata della Routing Table E_l viene estesa tramite l'aggiunta del seguente campo:

- ST: contiene l'informazione di digest, ovvero i valori presenti nei peer associati al sotto-albero radicato nel nodo figlio.

Da notare come le informazioni memorizzate siano dei riferimenti diretti ad indirizzi di rete dei peer. Questa è una scelta progettuale che consente di evitare un ulteriore livello di indirezione rappresentato dal passaggio per la rete DHT. In altri termini vengono velocizzate le operazioni di comunicazioni evitando inutili ricerche nella rete DHT.

4.1.3 Le operazioni

Nei due paragrafi seguenti saranno esaminate le operazioni di join e di ricerca effettuate da un nodo fisico in XCone.

Operazione di join

Per descrivere l'operazione di *join* di un nodo sulla rete XCone consideriamo l'esempio in Figura 4.2. Il peer P_0 esegue l'operazione di join nella rete DHT e in questa fase ottiene l'assegnamento dell'ID e del relativo nodo logico foglia

di XCone. Contestualmente all'ingresso in rete, la DHT invia al peer P_0 un riferimento ad un nodo già presente in rete, nell'esempio al nodo P_1, in modo tale da condividere con P_0 il prefisso più lungo dell'ID. Formalmente P_1 viene definito il *Least Common Ancestor (LCA)* di P_0.

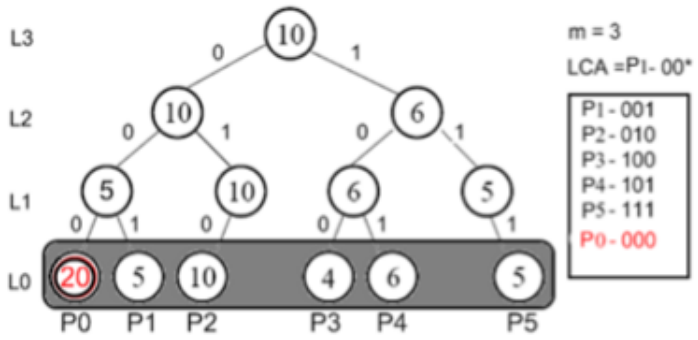


Figura 4.2: Join di un peer nella rete XCone

Prima di continuare con l'esempio descriviamo più in dettaglio il concetto di LCA e la sua localizzazione. In generale quando un peer P si unisce alla rete XCone ottiene mediante la DHT un identificatore ID. Supponiamo che al momento dell'entrata di P nel sistema, sulla DHT e quindi in XCone, siano già presenti N nodi con identificatori $ID_1, \ldots, ID_i, \ldots ID_N$. E' necessario che P individui tra questi nodi un nodo M il cui identificatore ID_m possieda la più lunga porzione di prefisso in comune con ID. E' possibile che esistano diversi nodi con questa caratteristica. Il nodo LCA può essere individuato mediante la DHT, infatti è possibile dimostrare che il predecessore o il successore di P sulla DHT è uno dei nodi il cui identificatore condivide con ID il più lungo prefisso. L'operazione di *lookup*, implementata in ognuna delle DHT attualmente esistenti, consente di reperire il successore ed il predecessore di un nodo e può essere utilizzata per supportare la ricerca del LCA in XCone. Il peer P ottiene dunque tramite la DHT il riferimento al suo predecessore ed al suo successore e sceglie tra questi quello con cui condivide il più lungo prefisso dell'identificativo.

Si consideri, ad esempio, la Figura 4.3. Si può notare come il nodo $N1$ sia quello che condivide il più lungo prefisso con $N2$, mentre il nodo $N3$, pur essendo adiacente ad $N2$ risiede dalla parte opposta all'albero XCone e pertanto non condivide alcun prefisso con $N1$.

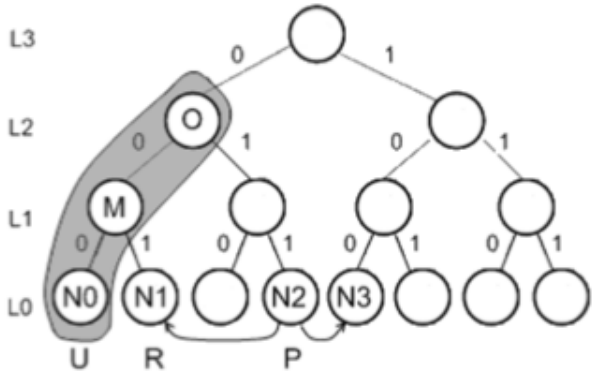


Figura 4.3: Localizzazione LCA

Una volta individuato il peer LCA, viene calcolato il nodo logico di intersezione tra i due cammini, nell'esempio rappresentato dal nodo O. Il nodo P invia un messaggio di join ad $N1$ specificando anche il livello L2 del nodo di intersezione O. Questo ulteriore parametro viene inserito in quanto, come nell'esempio riportato in Figura 4.3, il peer R a cui è associato il nodo logico $N1$, a seguito di un nuovo mapping dei nodi, potrebbe non avere più in gestione il nodo logico O.

La fase di *join* individua, attraverso la DHT, il nodo LCA con il procedimento descritto precedentemente. In seguito, attraverso il calcolo del nodo logico di intersezione, il nodo fisico LCA provvede a verificare se risulta ancora essere il gestore del nodo logico di intersezione e, nel caso in cui non lo fosse, provvede ad inoltrare la richiesta di join al proprio nodo padre. Questo procedimento esegue una risalita dell'albero lungo il cammino che porta al peer avente in gestione il nodo logico di intersezione. A questo punto l'operazione di *join* può continuare regolarmente con le ulteriori operazioni necessarie.

Riprendendo l'esempio iniziale riportato in Figura 4.2, sono stati indicati in basso i peer assegnati ai rispettivi nodi logici foglia, mentre all'interno di

ogni nodo logico è riportata la chiave gestita dal peer a cui il nodo logico è stato associato. Una volta assegnato il nuovo ID ed identificato il peer LCA, ha inizio la fase di risalita bit a bit o di *trickling*.

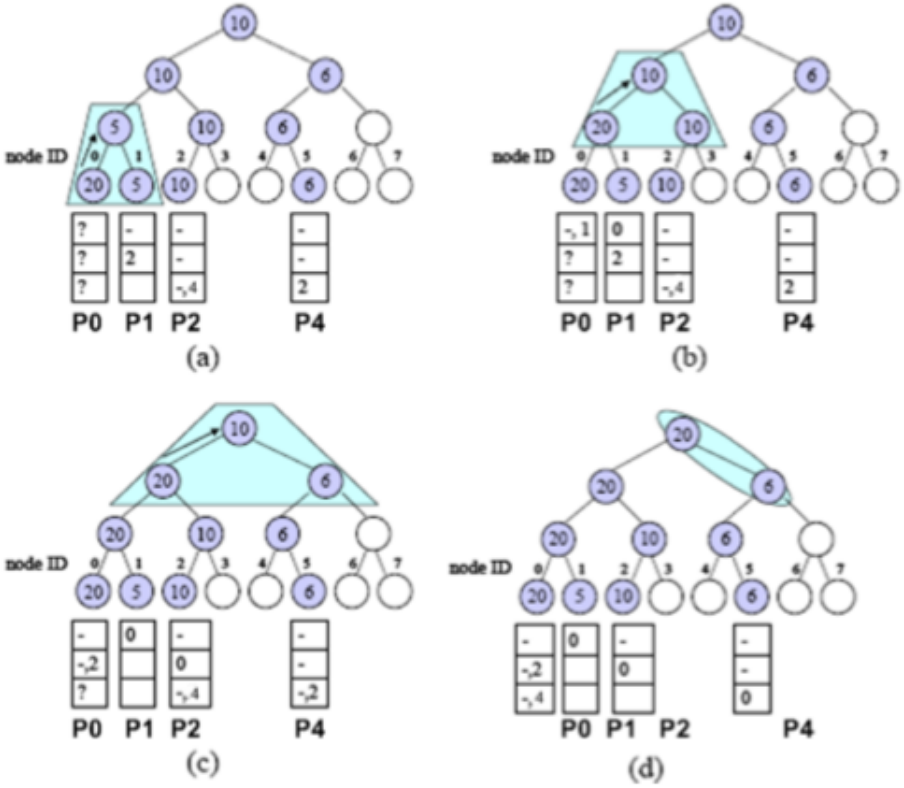


Figura 4.4: Fase di *trickling* in XCone

P_0 confronta il proprio ID con quello di P_1 determinando in questo modo il livello di partenza, $l = 1$ (Figura 4.4(a)). P_0 possiede un valore maggiore rispetto a quello gestito da P_1 e pertanto procede con l'inserire nella propria tabella di routing l'entrata $(-, IP_{P1}, d(P_1))$ (Figura 4.4(b)). Contestualmente viene informato anche il nodo P_1 sul nuovo padre. P_1 aggiornerà di conseguenza la sua routing table, inserendo al livello 1 l'entrata (IP_{P0}) e rimuovendo le informazioni ai livelli successivi (Figura 4.4(b) e (c)).

Continuando la risalita al livello 2, P_0 determina il nodo con cui confrontarsi grazie alle informazioni recuperate dalla tabella di routing del nodo P_1. Si

effettua un confronto con P_2 e anche in questo caso il nodo P_0 risulta essere il padre del livello 2. P_0 procede esattamente nello stesso modo aggiornando la tabella di routing al livello 2 con il valore $(-, IP_{P2}, D(P_2))$ ed informando P_2 sul cambio di padre (Figura 4.4(c)). La fase di *trickling* termina al terzo livello rilevando nuovamente un aggiornamento dell'albero e quindi una variazione nelle routing tables (Figura 4.4(d)).

Analizziamo adesso la complessità dell'operazione di join di un nodo nella rete XCone. La fase di inserimento nella DHT e di ricerca del nodo LCA viene effettuata dal livello sottostante con costo al caso pessimo di $O(\log T)$, con T numero massimo di peer. La fase di *trickling*, come visto nell'esempio, può impiegare al caso pessimo $O(\log T)$ operazioni pari al numero di peer presenti nel percorso dalla foglia alla radice dell'albero. L'operazione di *join* risulta dunque avere un costo complessivo logaritmico nel numero di peer.

Operazione di ricerca (find)

L'operazione di ricerca può essere eseguita a partire da un qualsiasi peer P della rete. Consideriamo il caso in cui un peer P voglia effettuare una query Q, in cui richiede K nodi con valore X compreso nell'intervallo $V \leq X \leq S$. La ricerca consiste in un'esplorazione dell'albero XCone mediante le informazioni contenute nella tabella di routing dei vari peer. La procedura di ricerca può essere suddivisa in due fasi. La prima fase consiste nella risalita dell'albero XCone denominata anche operazione di risalita o di *trickling*, mentre la seconda fase nell'esplorazione dei sotto-alberi. In XCone la prima operazione viene effettuata in maniera sequenziale, in particolare l'esplorazione di ogni sotto-albero, rappresentato da un'entrata della tabella di routing, viene eseguita controllando mano a mano che il numero di risorse localizzate sia sufficiente e in tal caso il procedimento di risalita viene interrotto. Nel processo di esplorazione dei sotto-alberi la ricerca avviene in parallelo guidata dalle informazioni di digest: dopo aver verificato tramite l'informazione di digest la presenza di valori in grado soddisfare i vincoli imposti dalla query, un nodo propaga immediatamente la richiesta di esplorazione in basso verso i nodi foglia.

Supponiamo che il generico peer P riceva una query Q, in cui si richiedono K risorse e che la query sia stata originata dal peer denominato *QueryNode*. Il generico peer P utilizzando una funzione *matchValue* controlla se la propria chiave soddisfa la query Q. In caso affermativo il nodo P invia il proprio indirizzo come risultato al *QueryNode*. Il *QueryNode* tiene traccia dei risultati raccolti e provvede a trasmettere al peer P il numero di peer che attualmente soddisfano i vincoli imposti dalla query Q. Il peer P pertanto analizza il valore ricevuto dal *QueryNode*, *collectedMatches*, e, nel momento in cui viene raggiunto il numero di risorse K richieste, termina l'operazione attraverso un'uscita forzata. Supponendo che le risorse localizzate non siano sufficienti, il peer P procede a collezionare un insieme S di propri sotto-alberi attraverso le informazioni possedute nella propria routing table. La funzione di digest consente di selezionare esclusivamente i sotto-alberi in cui è stata rilevata la presenza di risorse compatibili con la query Q. Per tali sotto-alberi, in maniera sequenziale, il nodo P invia una richiesta di esplorazione al peer che gestisce la radice del sotto-albero. Una volta inviato il messaggio, prima di passare alla successiva esplorazione, il peer P attende di ricevere una risposta dal *QueryNode* sul numero di risorse localizzate nella precedente richiesta. In questo modo, se il numero di risorse dovesse risultare sufficiente, viene interrotta la fase di esplorazione dei successivi sotto-alberi e di conseguenza terminata la fase di *trickling* attraverso un'uscita forzata. Questa operazione di sospensione dell'esplorazione ha come fine quello di ridurre al numero strettamente necessario il numero di nodi esplorati nell'albero XCone. Terminata la fase di esplorazione il peer P controlla se non sono stati esplorati sotto-alberi, in questo caso effettua direttamente l'operazione di *trickling* inviando un messaggio di richiesta al proprio nodo padre. Nel caso in cui siano stati esplorati i sotto-alberi, il nodo P sospende la ricerca inviando un messaggio di *trickling* al *QueryNode* ed attende nuovamente come risposta il numero di risorse localizzate. Anche in questo caso la sospensione è effettuata al fine di controllare la propagazione verso il nodo radice delle operazioni di ricerca. Per quanto riguarda le operazioni di esplorazione dei sotto-alberi, il nodo radice R controlla preliminarmente se la propria chiave soddisfa la query Q. In tal caso invia il proprio indirizzo al *QueryNode* e successivamente riceve

da quest'ultimo il numero di risorse localizzate. Nel caso in cui le risorse non siano sufficienti, R inoltra in parallelo le richieste di esplorazione dei propri sotto-alberi. La procedura di esplorazione procede verso i nodi logici foglia dell'albero XCone e viene diretta ai soli nodi foglia che, attraverso le informazioni di digest, risultano compatibili con la query Q. L'algoritmo procede nella maniera più veloce a recuperare le relative risorse.

Il costo di una operazione di ricerca in XCone è determinata dal costo della fase di *trickling*. Tale fase, come osservato, può coinvolgere nel caso pessimo un intero cammino dalla foglia al nodo radice dell'albero, pertanto al massimo $O(\log T)$ nodi fisici, con T numero di peer nella rete. Il numero totale di messaggi ricevuti dal *QueryNode*, avendo una esplorazione sequenziale dei sotto-alberi, è dato da al più K messaggi, corrispondenti ai risultati ricevuti, a cui vanno aggiunti $\log T$ messaggi di richieste di trickling per un totale di $O(\log T + K)$ messaggi ricevuti.

4.2 HASP

HASP estende XCone al caso multidimensionale. HASP è basato sull'uso di curve space-filling per la generazione di una chiave derivata a partire dal valore di n attributi che definiscono la singola risorsa.

Le curve space-filling sono curve continue che attraversano tutto lo spazio n-dimensionale delle risorse e permettono di identificare in modo univoco ciascuna n-upla di coordinate linearizzando uno spazio n-dimensionale e associando un valore univoco a ciascun punto di tale spazio. Tale valore prende il nome di *chiave surrogata* o *chiave derivata*.

Per riuscire ad eseguire range query multidimensionali su curve space filling è necessario mappare lo spazio multidimensionale dell'iper-rettangolo definito dalla range query nello spazio lineare definito dalla curva. Un approccio statico è quello di determinare tutti i segmenti di curva che sono contenuti nell'iper-rettangolo definito dalla range query senza considerare la distribuzione dei dati sulla curva space filling. Il problema principale di questo

approccio è l'elevato numero di segmenti che vengono generati al crescere del numero delle dimensioni e del range di valori che gli attributi possono assumere. Un secondo approccio, *guidato dai dati,* può essere realizzato secondo due diverse strategie. Una prima strategia consiste nel partire dalla query, individuare gli iper-quadranti che la intersecano e raffinare solo gli iper-quadranti che contengono dei dati. Il controllo che un iper-quadrante contenga dei dati può essere effettuato intersecando il segmento di curva corrispondente a quel quadrante con gli intervalli contenuti nel *digest.* Solo se l'intersezione tra il segmento di curva relativo ad un iper-quadrante che interseca la query ed il digest è diversa da zero, allora si procede al raffinamento. La seconda strategia parte invece dai segmenti contenuti nel digest, trova gli iper-quadranti che ricoprono tali segmenti e ne effettua quindi l'intersezione con la range query. In questo caso, si hanno a disposizione un insieme di intervalli di curva in cui sicuramente sono presenti dei dati e si deve passare da tali intervalli agli iper-quadranti che li contengono per poi intersecare questi quadranti con l'iper-rettangolo che descrive la range query.

E' facile intuire come la bontà dell'approssimazione restituita dalle tecniche di digest ricopra un ruolo fondamentale nella risoluzione di una range query multidimensionali in HASP.

4.3 Aggregazione di chiavi derivate: strutture di digest

La precedente versione di HASP includeva due tecniche di digest: i Bit Vectors Indexes ed i Q-Digest.

La tecnica BitVector Indexes definisce un vettore binario di k bits in maniera tale da catturare la distribuzione dei valori di un attributo A definito in un intervallo $[a, b]$. Il numero k degli intervalli di separazione rappresenta il numero di elementi del BitVector in cui ogni elemento indica se almeno una chiave derivata appartiene a tale intervallo o meno.

La tecnica Q-Digest sfrutta le proprietà matematiche dei quantili, per condensare una distribuzione di valori entro un margine di errore. Definito un insieme di valori n con dominio $[1, z]$, Q-Digest definisce un albero T di altezza $\log z$ in cui ogni livello partiziona l'intero dominio attraverso l'utilizzo di range $[min, max]$ assegnati ad ogni nodo.

Questa tesi ha introdotto in HASP due utleriori tecniche sofisticate come Wavelet e Bloom Filters.

4.3.1 Aggregazione mediante Wavelet

Grazie alle wavelets è possibile costruire un istogramma basato sulla distribuzione dei dati. Tale istogramma viene quindi utilizzato come funzione di digest per stimare l'ampiezza di range query.

Creazione dell'istogramma

Come descritto nel capitolo 3 la creazione dell'istogramma può essere divisa in tre fasi:

- dati i valori in input, calcola la distribuzione dei dati estesa T^{C+};
- una volta ottenuto T^{C+} esegue la wavelet decomposition utilizzando la funzione Haar wavelet;
- in base ai details coefficients ottenuti crea l'istogramma H.

Il calcolo della distribuzione dei dati estesa T^{C+} procede come segue:

Per ogni valore compreso tra il minimo ed il massimo la variabile che tiene traccia delle frequenze cumulate viene aggiornata e memorizzata all'interno dell'array delle frequenze cumulate S; se il minimo è maggiore di zero, per ogni valore compreso tra zero ed il minimo nell'array S viene inserita un'entry con valore zero.

Attraverso il processo sopra descritto se, ad esempio, in input avessimo i valori: $\{3, 7, 10, 15\}$ l'arrray di frequenze cumulate risulterebbe così composto:

$$S = \{0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4\}$$

possiamo notare come per i valori compresi tra zero ed il minimo (in questo caso 3) siano state inserite frequenze cumulate zero per rispecchiare il fatto che tali valori non sono presenti nell'insieme dei dati in input; successivamente per ogni valore compreso tra 3 e 15 è stata inserita la relativa frequenza cumulata.

Ottenuto l'array con la distribuzione dei dati estesa, si procede con la seconda e terza fase: calcolo delle wavelet decomposition utilizzando la funzione Haar wavelet e creazione dell'istogramma H.

Per ottenere la decomposizione wavelet l'array S delle frequenze cumulate che, alla prima iterazione, rappresenta i dati alla risoluzione massima viene scorso iterativamente. Ad ogni iterazione si calcolano le medie coppia a coppia delle frequenze cumulate e i detail coefficients; ottenuti questi valori l'array S è rimpiazzato con il risultato del calcolo delle medie mentre i details coefficients diversi da zero vengono inseriti in modalità bottom-up nell'istogramma H. Tale procedura prosegue finché l'array S delle frequenze cumulate non ha dimensione uno. All'ultima iterazione viene aggiunto, nella prima posizione di H, l'unico valore presente in S (risultato della media complessiva delle frequenze cumulate).

A questo punto il calcolo delle wavelet decomposition termina, l'istogramma H viene rappresentato come un array monodimensionale di lunghezza m, in cui ogni entry ha un coefficiente wavelet v_j ed un indice i_j.

$$H[j] = (v_j, i_j), \qquad 1 \leq j \leq m.$$

La procedura sopra descritta non possiede alcuna ottimizzazione riguardo l'occupazione di memoria.

Per insiemi di valori relativamente piccoli questa limitazione non rappresenta un problema ma, utilizzando le chiavi derivate con un range di valori $\left[0; 2^{n \times k}\right]$ con n uguale al numero di attributi e k pari al numero di bit da utilizzare per la codifica binaria del valore di ciascun attributo, l'occupazione di memoria diventa un aspetto critico al punto tale che la creazione dell'istogramma H diventa non applicabile. Ad esempio con $n = 6$ attributi e $k = 10$ le chiavi derivate possono assumere valori che partono da 9437188 fino ad arrivare a 13631559 con intervalli nell'ordine delle migliaia tra una chiave derivata e l'altra. In particolare sono state individuate due criticità che saranno oggetto di ottimizzazioni:

1. se il valore minimo è maggiore di zero, vengono inserite tante frequenze 0 per quanti sono i valori tra zero ed il minimo;
2. se nella distribuzione dei dati in input l'ampiezza dell'intervallo tra un valore ed il successivo è molto elevata la stessa frequenza viene inserita nell'array un numero elevato di volte;

Per affrontare le due criticità sopra descritte è stato deciso di modificare il modo in cui le frequenze cumulate vengono memorizzate nelle strutture dati. Di conseguenza viene proposta un'ottimizzazione della procedura per il calcolo delle wavelet decomposition.

In particolare, le frequenze cumulate non vengono più rappresentate da singoli valori ma da coppie della forma *(frequenza, count)*, dove *frequenza* è il valore della frequenza cumulata e *count* è il numero di volte che tale frequenza si ripete nella distribuzione. Adottando questa strategia l'utilizzo della memoria è diminuito drasticamente mantenendo tutte le informazioni necessarie.

Il seguente esempio mostra le modifiche proposte.

Esempio. Consideriamo un attributo X con dominio $D = [0, 20]$ con valori $\{0, 4, 7, 15\}$.

Per prima cosa calcoliamo le frequenze dei valori appartenenti all'attributo ottenendo:

$$Frequenze : [(0, 1)\,(4, 1)\,(7, 1)\,(15, 1)]$$

A questo punto otteniamo la distribuzione dei dati T^C:

$$Frequenze\ cumulate : [(0,1)\,(4,2)\,(7,3)\,(15,4)]$$

Come descritto precedentemente possiamo ricavare la distribuzione dei dati estesa T^{C+} ponendo frequenza zero ai valori non presenti ottenendo così:

$$\begin{aligned} Frequenze\ cumulate\ estese : \quad & [(0,1)\,(1,1)\,(2,1)\,(3,1)\,(4,2)\,(5,2)\,(6,2) \\ & (7,3)\,(8,3)\,(9,3)\,(10,3)\,(11,3)\,(12,3) \\ & (13,3)\,(14,3)\,(15,4)] \end{aligned}$$

Possiamo notare come ad ogni valore sia stata assegnata una frequenza cumulata.

Per poter eseguire range query utilizzando le wavelet abbiamo la necessità di eseguire l'algoritmo di wavelet decomposition partendo dalle frequenze dalla distribuzione dei dati estesa T^{C+} calcolata sopra:

$$S = [1,1,1,1,2,2,2,3,3,3,3,3,3,3,3,4]$$

Applicando l'ottimizzazione proposta, tali frequenze cumulate vengono rappresentate mediante coppie *(frequenza,count)* in cui *frequenza* rappresenta la frequenza cumulata e *count* il numero di volte che tale frequenza si ripete.

L'array S calcolato sopra sarà così composto:

$$S = [(1,4)\,(2,3)\,(3,8)\,(4,1)]$$

Possiamo notare immediatamente il notevole risparmio di memoria ottenuto.

A questo punto possiamo procedere con la decomposizione wavelet opportunamente modificata.

Risoluzione	Media	Detail Coefficients
16	$[(1,4)(2,3)(3,8),(4,1)]$	
8	$[(1,2)(2,1)(2.5,1),(3,3)(3.5,1)]$	$[(0,2)(0,1)(-0.5,1),(0,3)(-0.5,1)]$
4	$[(1,1)(2.25,1)(3,1),(3.25,1)]$	$[(0,1)(-0.25,1),(0,1)(-0.25,1)]$
2	$[(1.625,1)(3.125,1)]$	$[(0.625,1)(-0.125,1)]$
1	$[(2.375,1)]$	$[(-0.75,1)]$

Dall'esempio si evince come la decomposizione wavelet, in particolare il calcolo delle medie e dei detial coefficients, debba trattare in maniera opportuna il campo *count* delle coppie rappresentanti la distribuzione dei dati estesa T^{C+}.

Anche i details coefficients sono rappresentati mediante coppie *(valore,count),* l'inserimento di tali coefficienti all'interno dell'istogramma H terrà in considerazione tale rappresentazione.

Ricordiamo che l'istogramma H viene rappresentato come un array monodimensionale di lunghezza m, in cui ogni entry ha un coefficiente wavelet v_j ed un indice i_j. Otteniamo quindi:

$$H: \begin{array}{l}[(2.375,0)(-0.75,1)(-0.625,2)(-0.125,3)\\ (-0.25,5)(-0.25,7)(-0.5,11)(-0.5,15)]\end{array}$$

Di seguito è mostrato lo pseudocodice della soluzione proposta.

L'algoritmo 4.1 è dedicato alla decomposizione wavelet ed al calcolo dei detail coefficients, per sfruttare la rappresentazione ottimizzata delle frequenze cumulate è necessario eseguire la decomposizione tenendo conto del *valore* della frequenza cumulata e del *count* che tale frequenza possiede.

Algoritmo 4.1 Calcolo Haar Wavelet

```
1 INPUT: averages array di coppie(value,count)
2 Let: calculatedAvg array di coppie(value,count)
3 Let: calculatedDetailsCoeff array di coppie(value,count)
4 Let: avg,det,nextAvg,nextDet coppie(value,count)
5 for each item in averages do
6    if item.count isEven then
7       avg = (item.value,item.count/2)
8       det = (0,item.count/2)
9    end if
10   else if item.count isOdd then
11      if item.count > 1 then
12         avg = (item.value,(item.count-1)/2)
13         det = (0,(item.count-1)/2)
14      end if
15      if ∃ succ(item)
16         Let: nextItem = succ(item)
17         nextAvg = (((item.value+nextItem.value)/2),1)
18         nextDet = (((item.value-nextItem.value)/2),1)
19         nextItem.count = nextItem.count-1
20      end if
21   end if
22   ADD (avg,nextAvg) to calculatedAvg
23   ADD (det,nextDet) to calculatedDetailsCoeff
24 end for
25 return (calculatedAvg,calculatedDetailsCoeff);
```

Esempio. Supponiamo di avere in input un array di frequenze cumulate $S = [(1,4)\,(2,3)\,(3,9)]$. L'algoritmo sopra descritto procede come segue:

- Per il primo item, $(1,4)$, count è pari vengono quindi create due nuove coppie. La prima $(1,2)$ rappresenta la media, mentre la seconda $(0,2)$ rapprenta il detail coefficient.

- Il secondo item $(2,3)$, possiede un count dispari e maggiore di 1. Anche in questo caso si creano due coppie, come per l'iterazione precedente, considerando però il count dell'item decrementato di 1, otteniamo quindi una media $(2,1)$ ed un detail coefficient $(0,1)$. A questo punto viene eseguito il calcolo con l'item successivo $(3,9)$ ottenendo quindi una media $(2.5,1)$ ed un detail coefficient $(-0.5,1)$. Infine si decrementa di

una unità il count dell'item $(3, 9)$ dato che un'occorrenza di questa frequenza cumulata è appena stata utilizzata.

- Il terzo item, diventato adesso $(3, 8)$ produce una media $(3, 4)$ ed un detail coefficient $(0, 4)$.
- Tutte le medie ed i detail coefficient creati saranno inseriti negli array corrispondenti.

■

L'algoritmo 4.2 prende in input un array di frequenze cumulate rappresentate come coppie $(frequenza, count)$, calcolate attraverso l'algoritmo 4.1 ed esegue iterativamente la decomposizione wavelet di tale array inserendo in un insieme H i detail coefficients diversi da zero ottenuti.

Algoritmo 4.2 Creazione Istogramma

```
1  INPUT: cumulativeFreq array di frequenze cumulate
2  Let:H a set
3  Let:(calculatedDetailsCoeff,calculatedAvg) = CalcoloHaarWavelet(
       cumulativeFreq)
4  while size(calculatedAvg) > 1 do
5     for all item in calculatedDetailsCoeff
6        if item != 0 then
7           H = H ∪ (item)
8        end if
9     end for
10    (calculatedDetailsCoeff,calculatedAvg) = CalcoloHaarWavelet(
          calculatedAvg)
11 end while
```

Merge di istogrammi

Nell'esempio sopra abbiamo visto l'utilizzo delle wavelet per la creazione di un istogramma nel caso stand-alone, cioè presi dei dati in input sono state calcolate le frequenze cumulate e di seguito abbiamo eseguito la decomposizione wavelet per ricavare l'istogramma *H*.

Vediamo adesso come integrare l'utilizzo delle wavelet nell'albero *HASP*. In *HASP* ad ogni foglia viene assegnato un singolo valore, eventualmente una

chiave derivata, sulla base di tale valore viene costruita l'informazione di digest. Con il metodo visto nel paragrafo precedente ogni nodo invia al padre l'informazione di digest calcolata, a questo punto il nodo padre crea il proprio digest in modo da rappresentare i dati appartenti al proprio sotto-albero.

Utilizzando le wavelet come strumento di digest è sorta la necessità di riuscire a creare un nuovo istogramma calcolando il merge degli istogrammi provenienti dai nodi figli.

La soluzione proposta è la seguente:

- ogni nodo foglia calcola il minimo della distribuzione cumulata e l'istogramma H attraverso la decomposizione wavelet;
- ogni nodo invia al proprio padre il valore minimo e H;
- ogni nodo padre calcola il nuovo minimo, attraverso gli istogrammi rievuti ricostruisce i valori approssimati appartenenti ai propri figli, ordina in maniera crescente tali valori e calcola le frequenze cumulate così da poter creare il nuovo istogramma H attraverso la decomposizione wavelet.

Di seguito è mostrato lo pseudocodice della soluzione proposta. Nell'algoritmo 4.3 il nodo padre riceve tutte le informazioni necessarie dai propri figli, utilizza l'agoritmo 4.4 per ricostruire i valori approssimati, una volta ottenuti tali valori procede con la creazione delle frequenze cumulate da passare come argomento all'algoritmo 4.2 per la creazione dell'istogramma.

Algoritmo 4.3 Unione Wavelet

```
1 INPUT: h1,min1 istogramma e minimo ricevuto da figlio1
2        h2,min2 istogramma e minimo ricevuto da figlio2
3 Let: apprValuesChild1 = EspandiIstogramma(h1,min1)
4 Let: apprValuesChild2 = EspandiIstogramma(h2,min2)
5 Let: values array di interi
6 Let: cumulativeFreq array di coppie(value,count)
7 Let: count = 1
8 values = Sort(apprValuesChild1,apprValuesChild2);
9 for each val in values do
10    add(count,nextVal-val) to cumulativeFreq
11    count++;
12 end for
13 CreazioneIstogramma(cumulativeFreq)
```

Algoritmo 4.4 Espandi Istogramma

```
1 INPUT: H istogramma, min valore minimo
2 Let:cumulativeFreq = FrequenzeCumulateDaIstogramma(H)
3 Let:apprValues array di interi
4 Let:i = 1
5 apprValues[0] = min
6 for each item in cumulativeFreq do
7    apprValues[i] = apprValues[i-1] + item.count
8    i = i + 1
9 end for
10 return apprValues
```

L'algoritmo 4.5 sfrutta le proprietà matematiche della Wavelet viste nel capitolo 3, in particolare il Lemma 3, per ricostruire mediante la struttura *Error Tree* le frequenze cumulate a partire dall'istogramma H.

Algoritmo 4.5 Frequenze cumulate da Istogramma

```
1 INPUT:H istogramma
2 Let:errorTree //rappresentazione gerarchica di H
3 Let:currentFreq coppia di tipo (value,count)
4 Let:cumulativeFrequencies array di coppie (value,count)
5 Let: i = 0
6 for each leaf in errorTree do
7    if(i == 0)
8       currentFreq.value = leaf[i].value
9       currentFreq.count = 1;
10   end if
11   else if(leaf[i].value == leaf[i-1].value)
12      currentFreq.count = currentFreq.count+1;
13   end if
14   else (leaf[i].value != leaf[i-1].value)
15      ADD currentFreq to cumulativeFrequencies
16      currentFreq.value = leaf[i].value
17      currentFreq.count = 1;
18   end if
19   i = i + 1
20 end for
21 return cumulativeFrequencies
```

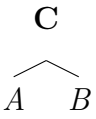
Esempio. Supponiamo che l'istogramma in input rappresenti i valori $5, 7, 10$. L'algoritmo sopra descritto procede come segue:

- Ricostruisce i valori delle frequenze cumulate a partire da H, ottenendo: $cumulativeFreq = [(1, 2)\,(2, 3)]$.
- Inserisce il valore minimo 5 all'interno dell'array di risultati.
- Analizza il primo item $(1, 2)$, inserisce nell'array di risultati un nuovo valore ottenuto come la somma tra il valore precedente nell'array di risultati ed il count dell'item, in questo caso $5 + 2 = 7$.
- Analizza il secondo item $(2, 3)$, inserisce nell'array di risultati un nuovo valore ottenuto come la somma tra il valore precedente nell'array di risultati ed il count dell'item, in questo caso $7 + 3 = 10$.
- Restituisce l'array dei risultati ottenuti: $[5, 7, 10]$.

■

Così facendo abbiamo la possibilità di fare il merge di due decomposizioni wavelet ottenendo un nuovo istogramma.

Esempio. Abbiamo un attributo X definito sul dominio $D = [0, 20]$ con valori $\{0, 4, 7, 15\}$, supponiamo di avere un nodo A con un istogramma che rappresenti i valori $\{7, 15\}$, un nodo B con un istogramma che rappresenti i valori $\{0, 4\}$ ed un nodo C, padre di A e B, che deve costruire un nuovo istogramma a partire dalle informazioni di digest provenienti da A e B.



- Nodo A:
  - Istogramma $H = [(1.5, 0)(-0.5, 1)]$, minimo $= 7$.
- Nodo B:
  - Istogramma $H = [(1.5, 0)(-0.5, 1)]$, minimo $= 0$.
- Nodo C:
  - Riceve da A: $H_A = [(1.5, 0)(-0.5, 1)]$, $min_A = 7$.
  - Riceve da B: $H_B = [(1.5, 0)(-0.5, 1)]$, $min_B = 0$.
  - Utilizza H_A per ricostruire le frequenze cumulate di A ottenendo $[(1, 8)\,(2, 1)]$; calcola i valori approssimati: $\{7, 15\}$.
  - Utilizza H_B per ricostruire le frequenze cumulate di B ottenendo $[(1, 4)\,(2, 1)]$; calcola i valori approssimati: $\{0, 4\}$.
  - Ordina i valori approssimati ottenuti in maniera crescente: $\{0, 4, 7, 15\}$.
  - Ottiene le frequenze cumulate: $S = [(1, 4)\,(2, 3)\,(3, 8)\,(4, 1)]$.
  - Crea l'istogramma:

$$H = \begin{array}{l}[(2.375, 0)(-0.75, 1)(-0.625, 2)(-0.125, 3) \\ (-0.25, 5)(-0.25, 7)(-0.5, 11)(-0.5, 15)]\end{array}$$

■

Utilizzando la soluzione proposta ogni nodo interno possiede tutte le informazioni necessarie per poter stimare, attraverso la struttura *error tree*, l'ampiezza di una range query nel proprio sotto-albero. In particolare il nodo radice C possiede esattamente lo stesso istogramma calcolato con il metodo stand-alone.

Fase di pruning

Una volta ottenuto l'istogramma eseguiamo la fase di pruning in modo da mantenere solamente un certo numero di coefficienti, in particolare quelli più significativi.

Nell'esempio precedente la decomposizione wavelet ha prodotto 8 coefficienti, supponiamo di voler mantenere solamente 4 coefficienti per ottimizzare l'uso della memoria e quindi poniamo $m = 4$.

La prima cosa da fare per poter scegliere i "migliori 4 coefficienti" è pesarli in qualche modo. In particolare per le Haar wavelet andiamo ad eseguire la normalizzazione dividendo i coefficienti: $\hat{S}(2^j), \ldots, \hat{S}(2^{j+1} - 1)$ per $\sqrt{2^j}$ per ogni $0 \leq j \leq \log N - 1$.

Come descritto nel capitolo 3 esistono diversi metodi, una volta eseguita la normalizzazione, per scegliere gli m coefficienti da mantenere; dal momento in cui utilizziamo le Haar wavelet che ricordiamo essere ortonormali, abbiamo scelto la strategia in cui manteniamo gli m coefficienti più grandi in valore assoluto.

Il numero di coefficienti da mantenere è proporzionale al numero di coefficienti presenti nell'istogramma.

Continuando con l'esempio precedente, il nostro istogramma con i coefficienti normalizzati risulterà come segue:

$$\begin{aligned} Normalized\ H &= [(2.375, 0)(-0.75, 1)(-0.4419, 2)(-0.0884, 3) \\ &\quad (-0.125, 5)(-0.125, 7)(-0.1768, 11)(-0.1768, 15)] \end{aligned}$$

A questo punto, eseguiamo la vera fase di pruning mantenendo solamente i 4 coefficienti più grandi in valore assoluto ed eliminando gli altri. Dopo la fase di pruning otteniamo:

$$\begin{aligned} Normalized\ Pruned\ H &= [(2.375, 0)(-0.75, 1) \\ &\quad (-0.4419, 2)(-0.1768, 11)] \end{aligned}$$

Abbiamo normalizzato H ed eseguito un pruning del 50% in modo da mantenere solamente i coefficienti più significativi, eseguiamo quindi la denormalizzazione in modo da ottenere i valori calcolati precedentemente e poter proseguire con la fase di query:

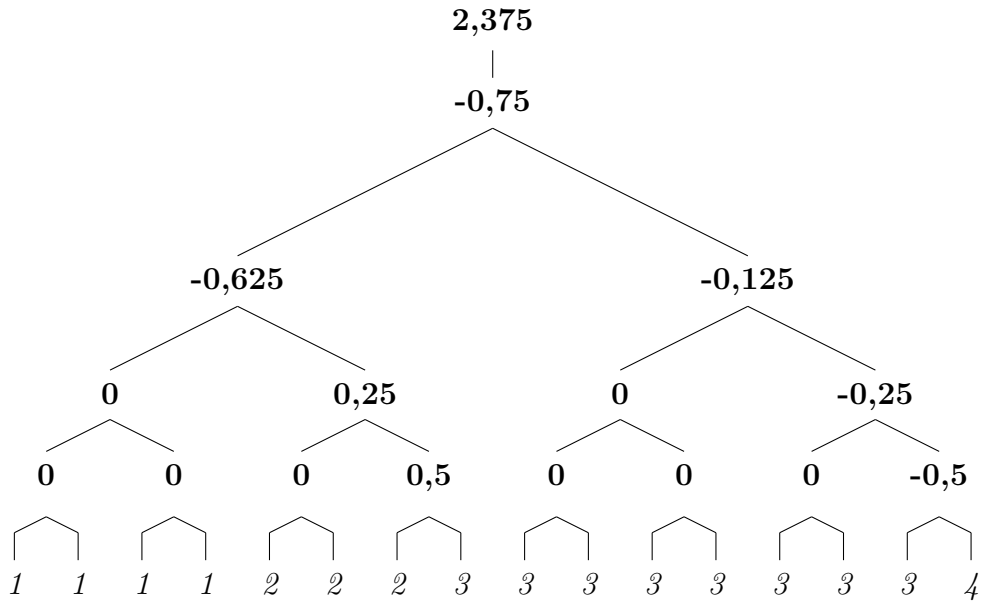
$$H = [(2.375, 0)(-0.75, 1)(-0.625, 2)(-0.5, 11)]$$

Range query

Nella fase di query è necessario stimare il risultato di una range query del tipo $l \leq X \leq h$. Come già visto nel capitolo 3 la strategia utilizzata è quella di ricostruire la frequenza cumulata di $l-1$ ed h, indicate rispettivamente con $S(l-1)$ e $S(h)$, utilizzando gli m coefficienti ottenuti grazie all'algoritmo di wavelet decomposition. L'ampiezza della query viene così stimata calcolando $S(h) - S(l-1)$.

La ricostruzione della frequenza cumulata di $l-1$ ed h avviene attraverso la struttura *error tree*.

Esempio. Di seguito è mostrata la struttura *error tree* rappresentante la decomposizione wavelet eseguita nell'esempio precedente, vediamo come con l'ausilio di tale struttura sia possibile ricavare i valori delle frequenze cumulate di $l-1$ ed h.



Ricordiamo che nell'albero i nodi interni rappresentano i details coefficient (risultato dell'algoritmo di wavelet decomposition) e la radice rappresenta la media globale dei valori in input, mentre le foglie, che vengono calcolate di volta in volta, rappresentano le frequenze cumulate.

Indichiamo con $S(i)$ il valore della foglia di indice i e con $\hat{S}(j)$ il valore di un nodo interno di indice j. Sfruttando le proprietà dell'*error tree* è possibile stimare l'ampiezza di una query $3 \leq x \leq 10$ calcolando i valori delle foglie $S(2)$ e $S(10)$:

$$S(2) = \hat{S}(0) + \hat{S}(1) - \hat{S}(2) + \hat{S}(5) - \hat{S}(9) = 2,375 - 0,75 + -0,625 + 0 - 0 = 1$$
$$S(10) = \hat{S}(0) - \hat{S}(1) + \hat{S}(3) - \hat{S}(6) + \hat{S}(13) = 2,375 + 0,75 - 0,125 - 1 + 0 = 3$$

La query $3 \leq x \leq 10$ viene quindi stimata con risultato $S(10) - S(2) = 3 - 1 = 2$

4.3.2 Bloom Filters

Abbiamo introdotto in HASP, oltre alle Wavelet anche i Bloom Filters.

Allo scopo di arricchire le funzioni di digest presenti in HASP con i *Bloom FIlters,* abbiamo definito un vettore binario B di n bits e k funzioni hash indipendenti che producono un valore distribuito uniformemente nel range $[1, \ldots, n]$; oltre alle funzioni hash ed al vettore binario B è possibile indicare il numero di elementi che verranno inseriti nel *Bloom Filters* e la percentuale di falsi positivi attesa. L'implementazione è stata eseguita in modo che si possa creare un nuovo *Bloom Filters* fornendo solamente il numero di elementi che verranno aggiunti e la percentuale di falsi positivi desiderata, il vettore binario B ed il numero di funzioni hash verranno calcolati di conseguenza.

Il range di valori dello spazio delle chiavi derivate ($\left[0; 2^{n \times k}\right]$ con n numero di attributi e k pari al numero di bit da utilizzare per la codifica binaria di ciascun attributo), non influenza in nessuno modo i *Bloom Filters* e la loro applicazione.

L'utilizzo dei *Bloom Filters* nell'albero distribuito HASP risulta particolarmente intuitivo.

- L'operazione di inserimento di una chiave derivata avviene applicando ad essa le k funzioni hash, ogni funzione hash restituirà una posizione del vettore binario B la quale sarà impostata a 1.
- L'operazione di unione tra più *Bloom Filters* consiste nell'eseguire l'OR bit a bit del vettore binario B e nell'aggiornamento dei rispettivi attributi.
- L'operazione di find di una chiave derivata all'interno del *Bloom Filters* applica le k funzioni hash alla chiave derivata da cercare e controlla il valore, nel vettore binario B, delle k posizioni restituite.
- La stima di una range query avviene attraverso l'integrazione con l'architettura HASP. In particolare, con l'ausilio di HASP, otteniamo le chiavi derivate appartenenti alla range query; a questo punto per ogni

chiave derivata ottenuta eseguiamo un'operazione di find sul Bloom Filters.

Capitolo 5

Implementazione

In questo capitolo verrà presentata l'implementazione per la definizione di nuove strategie di digest introdotte in *HASP,* il tutto è stato implementato mediante l'uso del framework *Overlay Weaver*, descritto nella sezione successiva.

5.1 Framework Overlay Weaver

Overlay Weaver è un framework *"Open Source"* realizzato come lavoro di ricerca da [36] e reperibile gratuitamente in rete. Le caratteristiche principali del framework consistono nella elevata *modularità* e *strutturazione a livelli*. Questo consente di realizzare applicazioni di rete in grado di appoggiarsi a moduli esistenti e di ereditarne un'elevata semplicità di configurazione. Il forte disaccoppiamento del framework consente infatti di effettuare agevolmente delle variazioni del comportamento dell'applicazione. OW supporta diversi overlay strutturati tra cui Chord [10], Pastry [12], Tapestry [13], Kademlia [14] e Koorde. Inoltre fornisce, come applicazioni ad alto livello, una semplice shell DHT interattiva e una shell Multicast. Il framework mette a disposizione anche una serie di tools per lo sviluppo e per il debugging come l'emulatore di nodi, il generatore di scenari ed un visualizzatore grafico della topologia della rete.

5.1.1 Architettura del framework

L'architettura di *Overlay Weaver,* come già accennato, risulta fortemente strutturata a livelli ed in particolare possiamo individuare i seguenti livelli:

- Applicazioni
- Servizi ad alto livello
- Servizi di routing
- Servizi di memorizzazione

La Figura 5.1 riassume graficamente l'architettura e consente di poter osservare le relazioni di dipendenza tra i vari livelli. Il livello delle applicazioni utilizza esclusivamente i servizi disponibili ad alto livello, analogamente i servizi ad alto livello si poggiano sui servizi di routing e di memorizzazione. Per quanto riguarda il livello di routing, *Overlay Weaver* effettua un'ulteriore suddivisione basata sul concetto di *Key-based routing*, *KBR*, introdotto da [37] che generalizza le funzionalità delle principali DHT. In generale, in una rete strutturata, ogni nodo memorizza un sotto-insieme di collegamenti ad altri nodi e la scelta di tali nodi identifica la topologia della rete. Per effettuare la ricerca di una chiave, il modello *KBR* consente di indirizzare la ricerca verso dei nodi via via sempre più vicini al nodo *target* in accordo ad una *metrica*. *Overlay Weaver* basandosi su tale modello suddivide ulteriormente il livello di routing in:

- ***Routing Driver,*** esponde le principali operazioni basste sul modello *KBR.*
- ***Routing Algorithm,*** contiene le diverse strategie di routing.
- ***Messaging Service,*** contiene diverse strategie di comunicazione di rete.

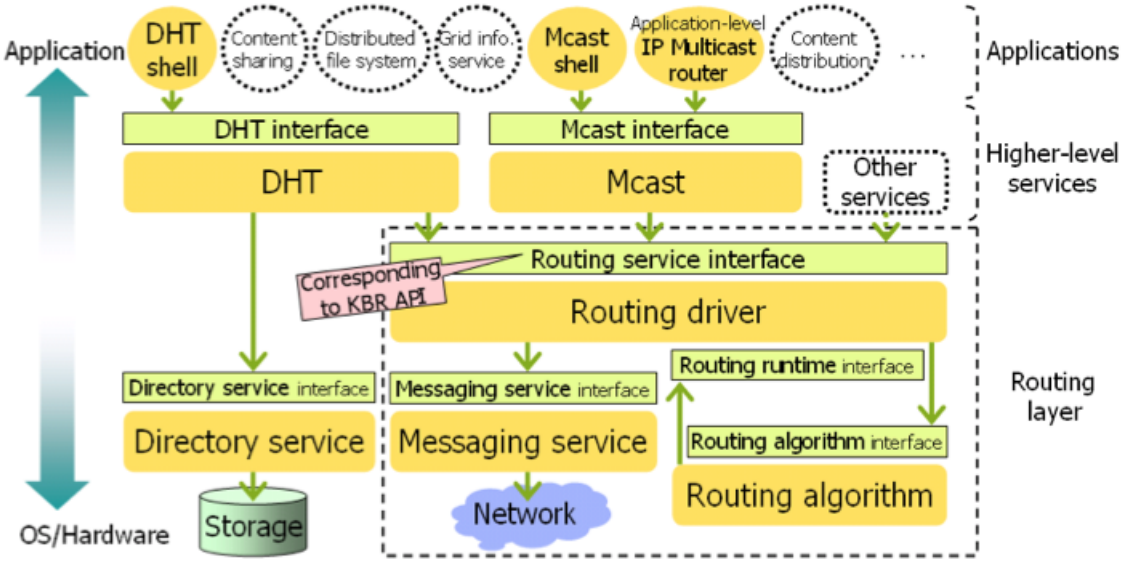


Figura 5.1: Architettura Overlay Weaver

Come implementazione del livello di *Routing Driver*, *Overlay Weaver* fornisce due versioni, una iterativa ed una ricorsiva. La Figura 5.2 illustra le diverse comunicazioni effettuate dai diversi tipi di routing.

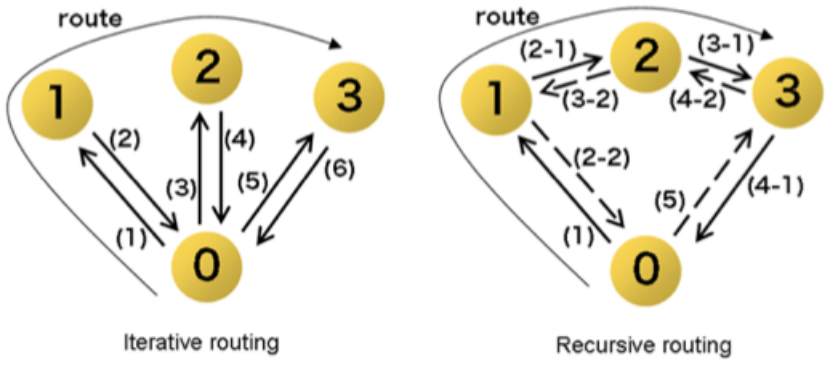


Figura 5.2: Tipologie di routing in Overlay Weaver

Le linee identificano i messaggi scambiati tra i vari nodi mentre i numeri annotati l'ordine di scambio. La notazione tratteggiata sta ad indicare i

messaggi che possono non essere necessariamente scambiati durante il routing, ma che servono in caso di utilizzo di protocolli non affidabili, quale ad esempio UDP. Per il livello di *Routing Algorithm*, *OW* implementa le strategie di routing Chord, Pastry, Tapestry, Kademlia e Koorde. Nel livello di *Messaging Service* sono implementate le classi per la comunicazione di rete attraverso i protocolli TCP, UDP e intra-thread utilizzato nel caso di emulazione. Nel livello di memorizzazione *OW* offre la memorizzazione attraverso un database relazionale (Berkeley DB) o in alternativa la memorizzazione in memoria principale attraverso le hash table della *Java standard class library*.

5.1.2 Tools di sviluppo

Overlay Weaver mette a disposizione una serie di strumenti di sviluppo aggiuntivi come l'emulatore di rete, il generatore di scenari e un visualizzatore grafico di rete. Lo strumento di maggior utilizzo risulta essere l'*emulatore*. Attraverso l'emulatore è possibile testare le applicazioni in maniera immediata in un ambiente controllato. L'emulatore prevede due modalità di esecuzione: la modalita locale in cui i nodi emulati sono localizzati nella sola macchina fisica locale e la modalita distribuita in cui varie istanze di emulatori sono attive in macchine fisiche distribuite e la comunicazione tra i rispettivi nodi emulati avviene attraverso la rete. Per quanto riguarda la modalità di interazione con lo strumento possiamo distinguere la modalità interattiva e la modalità batch. Nel primo caso i comandi sono impartiti all'emulatore attraverso una console testuale, mentre nel secondo caso viene fornito un file di testo o scenario contenente una serie di comandi in grado di essere interpretati ed eseguiti dall'emulatore.

La Figura 5.3 illustra la struttura tipica di uno scenario. In particolare è possibile notare le relative direttive in grado di eseguire una serie ripetuta di comandi e una sequenza di comandi in grado di temporizzare l'esecuzione. Infine la creazione di uno scenario può essere assistita da un ulteriore tool presente nel framework denominato generatore di scenari. L'ultimo strumento presente nel framework è il visualizzatore grafico di rete visibile in Figura

5.4. Il visualizzatore consente di rappresentare, in varie modalità grafiche, la locazione dei nodi nella rete e di visualizzare in tempo reale le comunicazioni tra i nodi. In particolare è possibile distinguere graficamente le varie tipologie di messaggi scambiati.

```
timeoffset 2000

#invoca il primo nodo
class.ow.tool.dhtshell.main
arg -p 10000
schedule 0 invoke

#invoca 3 nodi
arg
schedule 1000,1000,3 invoke

timeoffset 7000

#i 3 nodi entrano nell'overlay
schedule 0 control 1 init emu0
schedule 1000 control 2 init emu0
schedule 2000 control 3 init emu0

#inserisce la chiave di un nodo: put
schedule 4000 control 1 setdynamic key1 value1

#operazione di ricerca di una chiave: get
schedule 5000 control 2 getdynamic key1
```

Figura 5.3: Esempio di scenario interpretabile dall'emulatore

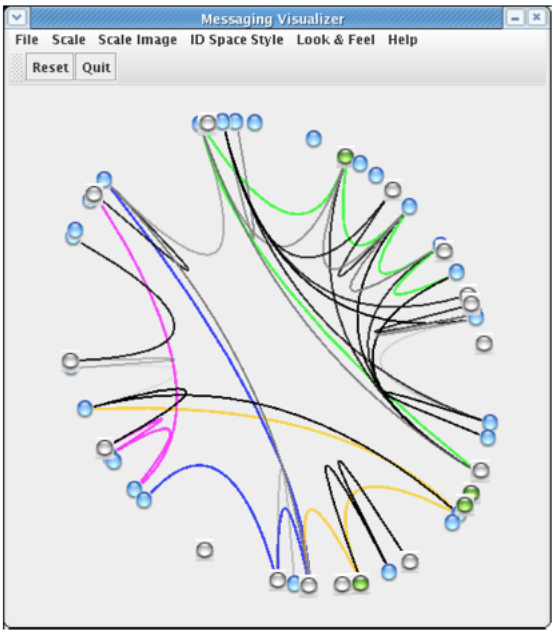


Figura 5.4: Esempio di visualizzazione grafica della rete

5.2 Definizione di uno scenario

L'emulazione di un numero N di nodi in locale necessita la definizione di un file di scenario contenente i vari comandi che l'interprete principale del sistema HASP deve eseguire. L'interprete principale del sistema è definito nella classe *ow.tool.xconeshell.main.java* e contiene la definizione dei vari comandi che possono essere invocati. Un file di scenario è composto da quattro sezioni principali:

1. definizione ed inizializzazione degli N nodi del sistema;
2. definizione dell'associazione delle risorse ai nodi del sistema;
3. fase di join dei nodi sulla rete;
4. definizione delle range query da eseguire sulla rete locale creata.

In Figura 5.5 è mostrata la prima parte di un possibile scenario. Si può notare come venga eseguito il comando *invoke* sul nodo 0, il nodo di *bootstrap* e poi, dopo 2 secondi da questo, su altri 99 nodi a distanza di 250ms l'uno dall'altro. Il comando invoke viene interpretato dall'interprete di *HASP* ed inizializza tutte le strutture dati di un nodo.

```
###################################
# SET-UP VIRTUAL NODES
###################################
# invokes the 1st XCone shell
class ow.tool.xconeshell.Main
schedule 0 invoke

# invokes others nodes
schedule 2000,250,99 invoke

# wait structures inizialization
timeoffset +5000
```

Figura 5.5: Scenario sezione 1, definizione ed inizializzazione dei nodi del sistema

In Figura 5.6 è mostrata la seconda sezione di un possibile scenario. In questa parte viene effettuata l'assegnazione delle risorse ai nodi mediante il comando *setdynamic*.

```
####################################
# SET-UP KEYS
####################################
# set dynamic attributes

schedule 0 control 0 setdynamic freeSwap 11 - cpuUse 2
schedule 1 control 1 setdynamic freeSwap 4 - cpuUse 7
schedule 2 control 2 setdynamic freeSwap 8 - cpuUse 0
schedule 3 control 3 setdynamic freeSwap 9 - cpuUse 4
schedule 4 control 4 setdynamic freeSwap 3 - cpuUse 4
schedule 5 control 5 setdynamic freeSwap 0 - cpuUse 14
schedule 6 control 6 setdynamic freeSwap 0 - cpuUse 4
schedule 7 control 7 setdynamic freeSwap 12 - cpuUse 14
schedule 8 control 8 setdynamic freeSwap 7 - cpuUse 2
schedule 9 control 9 setdynamic freeSwap 0 - cpuUse 11

# wait structures inizialization
timeoffset +5000
```

Figura 5.6: Scenario sezione 2, specifica ed assegnazione delle risorse ai nodi

In Figura 5.7 è mostrata la terza sezione di un possibile scenario nella quale viene effettuata la join dei nodi sulla rete HASP mediante il comando init. Il comando init esegue l'inserzione del nodo sulla DHT e nell'albero logico HASP.

```
####################################
# SET-UP XCONE OVERLAY
####################################
# start nodes join on DHT overlay

schedule    0 control 0 init emu0
schedule  500 control 1 init emu0
schedule 1000 control 2 init emu0
schedule 1250 control 3 init emu2
schedule 1500 control 4 init emu3
schedule 1750 control 5 init emu2
schedule 2000 control 6 init emu2
schedule 2250 control 7 init emu4
schedule 2500 control 8 init emu7
schedule 2750 control 9 init emu7

timeoffset +30000
```

Figura 5.7: Scenario sezione 3, *join* dei nodi sulla rete

In Figura 5.8 è mostrata la quarta ed ultima sezione di un possibile scenario. In questa parte vengono definite le range query da eseguire sulla rete HASP mediante il comando getdynamic. Il comando getdynamic assegna ad un nodo l'esecuzione di un range query.

```
###################################
# TESTS
###################################

schedule 0 control 0 getdynamic freeSwap 5:12 cpuUse 8:12 5
schedule 0 control 1 getdynamic freeSwap 1:6 cpuUse 4:10 2
schedule 0 control 2 getdynamic freeSwap 8:12 cpuUse 10:12 4
schedule 0 control 3 getdynamic freeSwap 2:7 cpuUse 3:6 7
schedule 0 control 4 getdynamic freeSwap 7:11 cpuUse 2:5 3
```

Figura 5.8: Scenario sezione 4, definizione delle range query

5.3 Implementazione tecniche di digest

L'elevata modularità di HASP ha permesso la definizione delle nuove strutture di digest presentate nel capitolo 4. Per ciascuna delle nuove tecniche di digest realizzate per il supporto a range query multidimensionali è stata necessaria l'implementazione dell'interfaccia *DigestInfo* e della classe astratta *DigestStrategy* presenti in HASP. L'interfaccia *DigestInfo* rappresenta l'informazione di digest utilizzata nelle operazioni in HASP e richiede che ciascuna tecnica di digest implementi i metodi:

- *public String getSerialized()*
- *public boolean equals(DigestInfo v1)*

Il metodo *getSerialized()* è necessario per la serializzazione dell'oggetto che contiene l'informazione di digest in modo che possa essere inviato ai vari nodi della rete.

La classe astratta *DigestStrategy* esprime le operazioni eseguibili su uno o più oggetti che contengono l'informazione di digest e che devono necessariamente implementare:

- *public abstract DigestInfo aggregate(DigestInfo v1, int level1, DigestInfo v2, int level2)*
- *public abstract int estimate(XConeQuery xQuery, DigestInfo value, int level)*

- *public abstract DigestInfo getDigestInfo(BigInteger value, int level)*
- *public abstract DigestInfo deserializeDigestInfo(String value)*

Il metodo *aggregate* è l'operazione necessaria affinchè le due informazioni di digest, provenienti da nodi differenti dell'albero XCone e ricevuti come parametro, possano essere unite in un'unico oggetto che contiene le informazioni di digest di entrambi.

Il metodo *estimate* riceve come parametri una range query e un'informazione di digest e restituisce il numero di risorse presenti nell'informazione di digest che soddisfano tale range query.

Il metodo *getDigestInfo* riceve come parametro la chiave del nodo e da questa crea l'oggetto che contiene l'informazione di digest.

Il metodo *deserializeDigestInfo* è il duale del metodo *getSerialized()* descritto in precedenza per l'interfaccia *DigestInfo*. Questo metodo riceve, come parametro in ingresso, una stringa risultato di un'operazione di serializzazione dell'informazione di digest di un altro nodo della rete e, a partire da tale stringa, ricostruisce l'oggetto che contiene l'informazione di digest di quel nodo.

Poichè ciascuna strategia di digest deve implementare l'interfaccia *DigestInfo* e la classe astratta *DigestStrategy* descritte nel paragrafo precedente, anche la tecnica di digest Wavelet deve farlo. Le classi che definiscono tale strategia di digest in HASP sono:

- *Wavelet:* contiene la rappresentazione della decomposizione wavelet attraverso un istogramma *H*. Tale istogramma è rappresentato mediante un *HashMap* che contiene informazioni sull'indice dell'istogramma e sul valore del detail coefficients a tale indice.
- *WaveletElement:* contiene la rappresentazione delle frequenze cumulate tramite coppie *(frequenza,valore)*.

La classe *WaveletStrategy* implementa la classe astratta *DigestStrategy.* Il metodo *aggregate* unisce due istogrammi *H* provenienti da nodi differenti

della rete, in questa nuova tecnica di digest è stato implementato l'algoritmo per la fusione di due wavelet decomposition così come presentato nella sezione . Il metodo *estimate* deve restituire il numero di chiavi derivate che soddisfano una determinata range query e contiene dunque al suo interno l'algoritmo del processo di risoluzione delle range query multidimensionali, attraverso la struttura *error tree*, descritto nella sezione .

La classe *Wavelet* implementa l'interfaccia *DigestInfo*. La serializzazione dell'istogramma H, rappresentante l'informazione di digest, avviene tramite la concatenazione di stringhe, nello specifico ciascuna informazione di digest viene serializzata nel seguente formato:

min sep max sep WaveletElement sep WaveletElement sep...

in cui *sep* indica un carattere separatore dei vari campi e *WaveletElement* indica la serializzazione della frequenza cumulata rappresentata mediante coppie *(frequenza, valore)*. Il processo di deserializzazione, a partire dalla stringa ricevuta, ricava i vari elementi effettuando un'operazione di split della stringa tramite il carattere separatore, con i dati ricevuti crea un nuovo oggetto *Wavelet* replicando così sul nodo destinatario la stessa informazione di digest presente nel nodo mittente. Il metodo *getDigestInfo* in questa strategia di digest costruisce uno nuovo oggetto *Wavelet*.

5.4 Codice degli algoritmi

In questa sezione è mostrato lo pseudocodice degli algoritmi descritti nel capitolo , rispettivamente per la creazione della distribuzione dei dati estesa, per il caclolo della decomposizione wavelet, per la popolazione dell'istogramma H e per la risoluzione di range query utilizzando la struttura *error tree*.

Algoritmo 5.1 Creazione della distribuzione dei dati estesa

```
1  /* values è un variabile di tipo ArrayList in cui
2   * sono memorizzati i valori in input per cui è
3   * necessario calcolare le frequenze cumulate
4   */
5  ArrayList<WaveletElement> S = new ArrayList<WaveletElement>();
6  for(int j=0; j<values.size()-1; j++)
7  {
8      WaveletElement element = new WaveletElement(j+1,values.get(j
           +1)-values.get(j));
9      initialCoefficients.add(element);
10 }
```

La decomposizione wavelet e la popolazione dell'istogramma H avvengono di pari passo. Ricordiamo che la decomposizione wavelet è un processo ricorsivo sulle frequenze cumulate; ad ogni iterazione l'algoritmo *makeUpCoefficientsAndAvarages(HashMap waveTrans)* prende in input un *HashMap* in cui all'interno sono memorizzate le medie ed i detail coefficients calcolati all'iterazione precedente e restituisce un nuovo *HashMap* con le medie ed i detail coefficients calcolati per quell'iterazione, a questo punto i detail coefficients diversi da zero vengono inseriti in una struttura rappresentante l'istogramma H.

Algoritmo 5.2 Decomposizione Wavelet

```
1  public HashMap makeUpCoefficientsAndAvarages(HashMap waveTrans)
2  {
3    /* Inizializza strutture dati */
4    ArrayList<WaveletElement> avgList = new ArrayList<WaveletElement>();
5    ArrayList<WaveletElement> detList = new ArrayList<WaveletElement>();
6    HashMap waveletTransform = new HashMap();
7
8    for(int i=0;i<waveTrans("avarages").size; i++)
9    {
10     WaveletElement currElement = waveTrans("avarages").get(i);
11     /* Count pari */
12     if(currElement.get(i).getCount()%2==0 && currElement.get(i).getCount()>0)
13     {
14       WaveletElement avg =
15           new WaveletElement(currElement.getValue(),currElement.getCount()/2);
16       WaveletElement det = new WaveletElement(0,currElement.getCount()/2);
17       avgList.add(avg);
18       detList.add(det);
19     }
20     /* Count dispari */
21     else
22     {
23       /* Caso in cui il count e' maggiore di uno */
24       if(currElement.getCount()>1)
25       {
26         /* Non e' l'ultimo elemento della lista */
27         if(i+1<waveTrans.get(avarages).size())
28         {
29           WaveletElement avg =
30               new WaveletElement(currElement.getValue(),(currElement.getCount()-1)/2);
31           WaveletElement det = new WaveletElement(0,(currElement.getCount()-1)/2);
32           avgList.add(avg);
33           detList.add(det);
34
35           /* Crea una nuova media ed un nuovo detail coefficients
36            * in collaborazione con l'elemento successivo
37            */
38           WaveletElement nextElement = waveTrans("avarages").get(i+1);
39           WaveletElement nextAvg =
40               new WaveletElement(currElement.getCount()+nextElement.getValue())/2,1);
41           WaveletElement nextDet =
42               new WaveletElement(currElement.getCount()nextElement.getValue())/2,1);
43           avgList.add(nextAvg);
44           detList.add(nextDet);
45
46           /* Aggiorna il count della frequenza successiva */
47           nextElement.setCount(nextElement.getCount()-1);
48         }
49         /* E' l'ultimo elemento della lista */
50         else
51         {
52           WaveletElement avg =
53               new WaveletElement(currElement.getValue(),currElement.getCount()/2);
54           WaveletElement det = new WaveletElement(0,currElement.getCount()/2);
55           avgList.add(avg);
56           detList.add(det);
57         }
58       }
59       /* Caso in cui il count e' uguale a uno */
60       else if(currElement.getCount()==1)
61       {
62         /* Se l'elemento è l'ultimo nella lista lo scartiamo, altrimenti calcola
63          * la media ed il detail coefficients in collaborazione con l'elemento
                successivo
64          */
65         if(i+1<waveTrans.get(avarages).size())
66         {
67           WaveletElement nextElement = WaveletElement("avarages").get(i+1);
68           WaveletElement avg =
69               new WaveletElement(currElement.getCount()+nextElement.getValue())/2,1);
70           WaveletElement det =
71               new WaveletElement(currElement.getCount()nextElement.getValue())/2,1);
72           avgList.add(avg);
73           detList.add(det);
74
75           /* Aggiorna il count della frequenza successiva */
76           nextCoeff.setCount(nextElement.getCount()-1);
77         }
78       }
79     }
80   }
81   /* Inserisce le medie ed i detail coefficients calcolati
82    * nell'HashMap
83    */
84   wavletTransform.put("avarages", averageCoefficients);
85   wavletTransform.put("detailCoefficients", detailCoefficient);
86   return wavletTransform;
87 }
```

Algoritmo 5.3 Creazione dell'istogramma H

```
1  Compute_WaveletDecomposition(Coeff [] cumFreq)
2  {
3      /* Istogramma H */
4      HashMap<Integer,double[]> H = new HashMap<Integer,double[]>();
5
6      /* Contatore per l'inserimento dei detail coffiecients in H */
7      int count = S.size()-1;
8
9      /* HashMap con medie e  detail coefficients */
10     HashMap<String,Array<WaveletElement>> waveletTransform =
11           new HashMap<String,Array<WaveletElement>>();
12     /* Inizialmente l'array detail coefficients è vuoto */
13     Array detailCoefficients = new Array();
14     wavletTransform.put("avarages", S);
15     wavletTransform.put("detailCoefficients", detailCoefficients);
16
17     while(wavletTransform("avarages").size() > 1)
18     {
19         wavletTransform = makeUpCoefficientsAndAvarages(wavletTransform);
20
21         for(i=wavletTransform.get("detailCoefficients").size-1; i>0; i--)
22         {
23             Coeff c = wavletTransform.get("detailCoefficients").get(i);
24             /* Inserisce in H solo i coefficienti diversi da 0 */
25             if(c.getValue()!=0)
26             {
27                 H.put(count,[count,c.getValue]);
28             }
29             count=count - c.getCount();
30         }
31
32         /* Inserisce in H l'unico valore nell'array */
33         H.put(count,(count,wavletTransform("avarages").get(0).getValue()));
34
35     }
36 }
```

Per stimare l'ampiezza di una range query della forma $l \leq x \leq h$ sfruttando l'istogramma H, utilizziamo il seguente algoritmo:

Algoritmo 5.4 Calcolo ampiezza range query

```
1  public double ComputeSelectivity(BigInteger low, BigInteger high)
2  {
3    double selectivity = 0;
4    double aggregationLow = 0;
5    double aggregationHigh = 0;
6
7    Collection<double[]> c = H.values();
8    Iterator<double[]>itr = c.iterator();
9    while(itr.hasNext())
10   {
11     double[]temp = (double[])itr.next();
12     if(temp != null){
13       if(Contribute(temp[0],low,range)){
14         aggregationLow =
15         aggregationLow + Compute_Contribution(temp[0],temp[1],low,range);
16       }
17       if(Contribute(temp[0],high,range)){
18         aggregationHigh =
19         aggregationHigh + Compute_Contribution(temp[0],temp[1],high,range);
20       }
21     }
22   }
23   selectivity = (aggregationHigh - aggregationLow);
24   return selectivity;
25   }
26 }
```

Il metodo *Contribute(i,j,range)* restituisce *true* se il coefficiente con indice i contribuisce alla ricostruzione del valore della foglia *S(j),* altrimenti restituisce *false.*

Il metodo *Compute_ Contribution(i,v,j,range)* calcola il contributo del coefficiente *(i,v)* alla ricostruzione della foglia *S(j).*

Algoritmo 5.5 Controlla se un nodo interno i contribuisce al calcolo della foglia *S(j)*

```
1  private boolean Contribute(double i, BigInteger j, BigInteger N)
2  {
3    /* Ricava l'indice della foglia piu' a sinistra del sotto-albero
4     * con radice in i e N foglie
5     */
6    BigInteger Ll = left_most_leaf(i, N);
7
8    /* Ricava l'indice della foglia piu' a destra del sotto-albero
9     * con radice in i e N foglie
10    */
11   BigInteger Lr = right_most_leaf(i,N);
12
13   /* Controlla se j appartiene al sotto-albero */
14   if((j >= Ll) && (j <= Lr))
15     return true;
16   else
17     return false;
18 }
```

Algoritmo 5.6 Calcola il contributo di un nodo interno i per ricostruire il valore della foglia *S(j)*

```
1  provate double Compute_Contribution(double i, double v, BigInteger j, BigInteger N)
2  {
3    double contribution = v;
4    BigInteger Lrl;
5    BigInteger Lrr;
6
7    /* Caso speciale, nodo radice */
8    if(i == 0)
9      return contribution;
10
11   /* Calcola  l'altezza del nodo i nell'error tree*/
12   double depth = Math.log((double)i)/Math.log(2);
13   double logN = ((Math.log(N.doubleValue()))/Math.log(2))-1;
14   //Node i is not a parent of leaves
15   /* Il nodo i è un nodo interno non padre di foglie */
16   if(depth < logN){
17     /* Calcola l'indice della foglia piu' a sinistra del
18      * sotto-albero radicato in i */
19     Lrl = left_most_leaf(right_child(i),N);
20     /* Calcola l'indice della foglia piu' a destra del
21      * sotto-albero radicato in i */
22     Lrr = right_most_leaf(right_child(i), N);
23   }
24   /* Il nodo i è padre di foglie */
25   else{
26     Lrl = right_most_leaf(i, N);
27     Lrr = Lrl;
28   }
29   if((j >= Lrl) && (j <= Lrr))
30     contribution = contribution*(-1);
31   return contribution;
32 }
```

Capitolo 6

Risultati sperimentali

Questo capitolo descrive la parte sperimentale della tesi. I test presentati in questo capitolo hanno come obiettivo la valutazione sperimentale dell'utillizzo delle Wavelet e dei Bloom Filters in un contesto di range queries su albero di aggregazione.

Nella sezione 6.1 verrano riportati i risultati sperimentali sull'effetto del pruning nelle wavelet su dataset sintetico. Nella sezione 6.2 verrà riportato il lavoro di tuning eseguito sui Bloom Filters in modo da individuare il giusto trade-off tra affidabilità e memoria utilizzata. Nella sezione 6.3 verrà descritto l'ambiente di test *PlanetLab* e la metodoligia di definizione dei test, successivamente verrà proposta una validazione delle wavelet con i dati *PlanetLab* ed infine verranno riportati i risultati del confronto tra le due tecniche di digest Wavelet e Bloom FIlters.

6.1 Valutazione wavelet

Come descritto nel capitolo 4, per quanto riguarda la tecnica di digest Wavelet, la strategia di pruning adottata permette di decidere la percentuale di coefficienti da eliminare. Attraverso l'applicazione del pruning otteniamo una minore occupazione di memoria ed un minor utilizzo di banda di trasmissione a discapito di un errore nella risoluzione di range quey. Lo scopo

di questa sezione è quello di valutare, al variare della percentuale di pruning, il risparmio di memoria ottenuto e l'errore nella risoluzione di range query introdotto così da trovare un trade-off soddisfacente.

Per l'esecuzione dei test è stato generato un albero con 1024 nodi. Ad ogni nodo è stato assegnata una chiave random generata secondo una distribuzione uniforme. Le foglie sono rappresentate a livello 0 mentre la radice dell'albero a livello 10. L'incremento della percentuale di pruning avviene con step del 10%.

Per ogni step di pruning viene mostrata:

- per ogni livello la media della dimensione wavelet di tutti i nodi appartenenti a quel livello;
- una media della dimensione wavelet di tutti i nodi presenti nell'albero.
- l'errore introdotto nella risoluzione di range query.

Le query sono state generate scegliendo il bound della query mediante una distribuzione uniforme.

Ricordiamo che per la definizione dell'errore abbiamo utilizzato le misure proposte nel capitolo 3. Utilizziamo l'*absolute error* per il calcolo dell'errore di ogni singola query e la *Norma 2* per il calcolo dell'errore globale.

Dalla Figura 6.1 è possibile notare come l'effetto del pruning sulla dimensione media delle wavelet aumenti mano a mano che saliamo di livello nell'albero fino ad avere nella radice l'effetto più rilevante. Questi risultati ci mostrano come l'occupazione di memoria in ogni singolo nodo e la banda di trasmissione occupata diminuiscano con l'aumentare della percentuale di pruning applicata.

	Pruning									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Livello 1	3	3	3	3	3	3	3	3	3	3
Livello 2	5	5	5	5	5	3,81	3,81	3,75	3	3
Livello 3	9	9	8,88	8,73	8,28	5,94	5,54	4,14	3	3
Livello 4	17	16,96	16,53	15,84	14,10	9,6	8,37	5,68	3,07	3
Livello 5	33	32,87	31,62	28,9	23,84	15,9	12,65	8	3,68	3
Livello 6	65	64,75	61,37	54,12	41,62	26,43	19,25	11,25	5,18	3
Livello 7	129	128,5	120,75	102,62	74	46,37	32,25	17,12	6,62	3
Livello 8	257	256	237,75	189,25	131	73,75	46	22,75	7,75	3
Livello 9	513	511	469	346,5	235	130	78	37	11,5	3
Livello 10	1025	1021	916	619	401	230	137	62	15	3
Media	205,6	204,808	186,99	137,296	93,684	54,48	34,587	17,469	6,18	3

Figura 6.1: Dimensione media wavelet al variare della percentuale di pruning adottata

Ad esempio il valore $5,66$ mostra la dimensione media dell'array contenente le coppie di frequenze cumulate a livello 4 con pruning del 70%.

Di seguito vediamo l'andamento della dimensione media delle wavelet di tutti i nodi dell'albero.

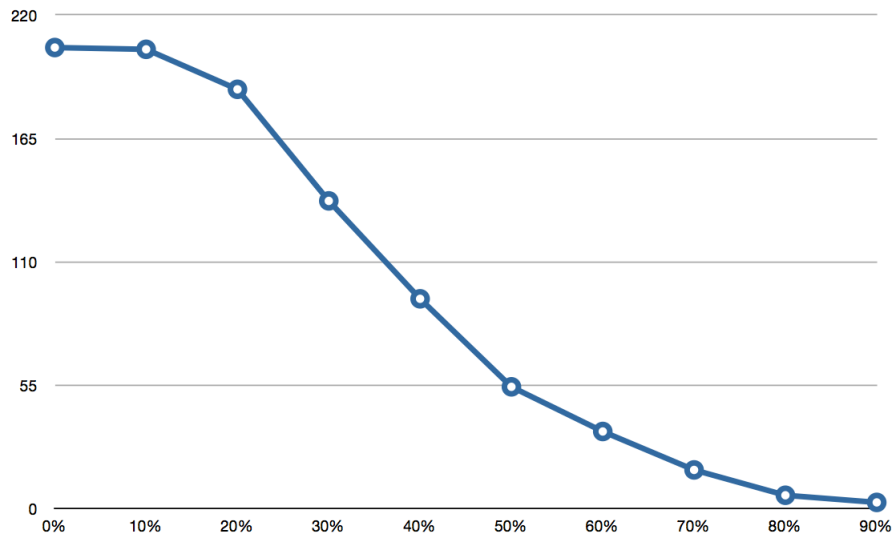


Figura 6.2: Dimensione media wavelet di tutti i nodi dell'albero al variare della percentuale di pruning adottata

Dalla Figura 6.3 è possibile notare come per tutti i livelli la media dell'errore globale aumenti all'aumentare della percentuale di pruning. Anche in questo caso l'errore è più rilevante mano a mano che saliamo di livello.

	Pruning									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Livello 1	0	0	0	0	0	0	0	0	0	0
Livello 2	0	0	0	0	0	0,004	0,004	0,007	0,007	0,007
Livello 3	0	0	0	0	0,002	0,011	0,012	0,013	0,014	0,014
Livello 4	0	0	0	0	0,011	0,020	0,024	0,024	0,027	0,031
Livello 5	0	0	0,01	0,013	0,022	0,041	0,042	0,042	0,057	0,063
Livello 6	0	0	0,019	0,027	0,044	0,071	0,098	0,126	0,115	0,126
Livello 7	0	0	0,055	0,039	0,055	0,167	0,230	0,230	0,230	0,253
Livello 8	0	0	0,111	0,176	0,335	0,447	0,506	0,506	0,506	0,506
Livello 9	0	0	0,223	0,353	0,670	0,894	1,012	1,012	1,012	1,012
Livello 10	0	0	0,447	0,707	1,341	1,788	2,024	2,024	2,024	2,024
Somma	0	0	0,865	1,315	2,48	3,443	3,952	3,984	3,992	4,036

Figura 6.3: Errore globale nella risoluzione di range query al variare della percentuale di pruning adottata

Di seguito vediamo l'andamento della somma degli errori in tutto l'albero al variare della percentuale di pruning.

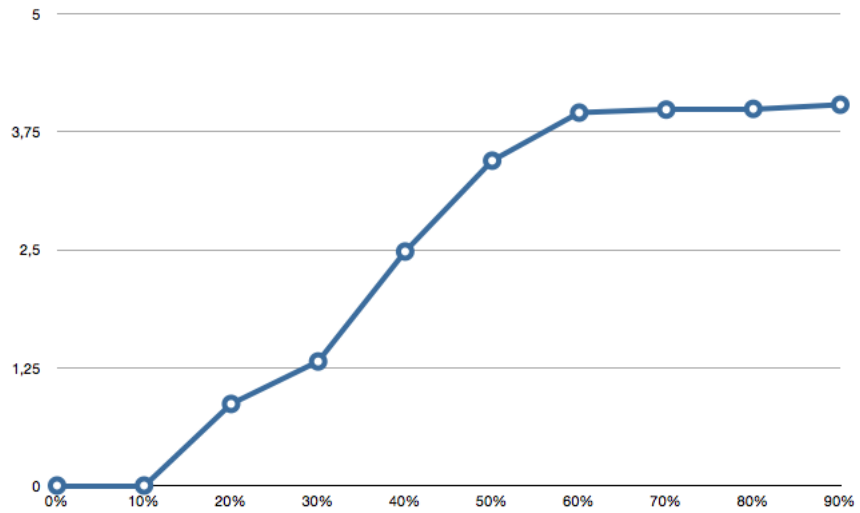


Figura 6.4: Somma degli errori di tutti i nodi dell'albero al variare della percentuale di pruning adottata

Questi risultati ci mostrano come l'occupazione di memoria in ogni singolo nodo e la banda di trasmissione occupata diminuiscano con l'aumentare della percentuale di pruning applicata, allo stesso tempo abbiamo un aumento dell'errore globale di ogni nodo nella risoluzione di range query.

Nel caso di distribuzione uniforme di chiavi e query un buon compromesso sembra essere un pruning del 30%, in quanto produce wavelet di dimensione contenuta ed un errore limitato.

6.2 Tuning Bloom Filters

Come descritto nel capitolo 4 la tecnica di digest Bloom Filters è stata progettata in modo che fosse possibile decidere la probabilità di falsi positivi desiderata. Dalle proprietà matematiche dei Bloom Filters si evince che una minor probabilità di falsi positivi comporta un maggior numero di fuzioni hash ed un vettore binario di dimensione più grande, in altre parole diminuendo i falsi positivi aumenta la memoria occupata e la computazione da eseguire.

Lo scopo di questo paragrafo è quello di eseguire un tuning in HASP per individuare il giusto trade-off tra memoria utilizzata ed affidabilità del Bloom Filters. In particolare eseguiamo i test sopra descritti con una percentuale di falsi positivi crescente.

Dal grafico in Figura 6.5 al grafico in Figura 6.7 è possibile notare come con l'aumentare della probabilità di falsi positivi aumenta la sovrastima media del numero di risorse trovare rispetto al valore y di risorse richieste, aumenta inoltre il traffico di rete dato che crescono il numero medio di nodi contattati e lo sbilanciamento tra il numero massimo ed il numero minimo di nodi contattati. Risulta interessante il grafico in Figura 6.8 il quale ci mostra che con l'aumentare della probabilità di falsi positivi diminuscono il numero medio di query che durante il processo di risoluzione di una query raggiungono la radice HASP. Questo risultato è dovuto al fatto che in HASP il nodo A, che genera la query, inoltra il messaggio di query risalendo l'albero. Inviato il

messaggio A rimane in attesa del numero di risorse che soddisfano la query. Una volta ricevuto il numero di risorse A decide se continuare con la risalita dell'albero o interrompere il processo di risoluzione query. Aumentando la probabilità di falsi positivi aumenta la sovrastima dell'informazione di digest ed allo stesso tempo diminuiscono i messaggi di risalita dell'albero HASP.

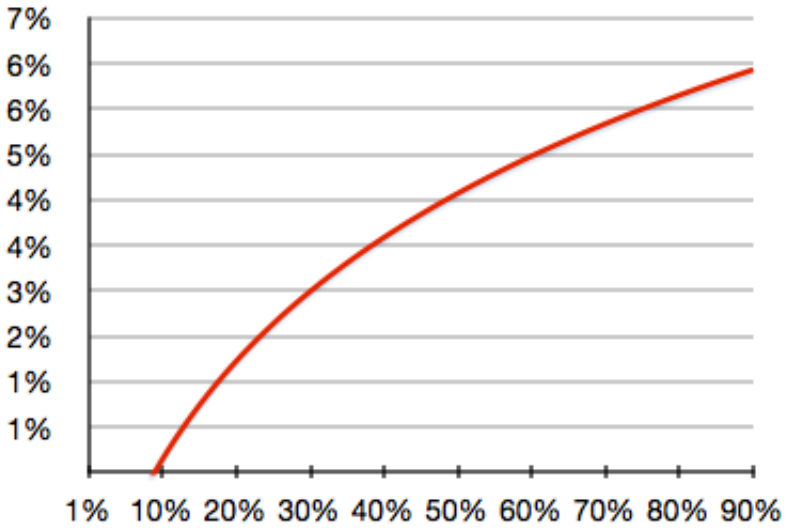


Figura 6.5: Sovrastima media del numero di risultati ricevuti rispetto al valore y richiesto

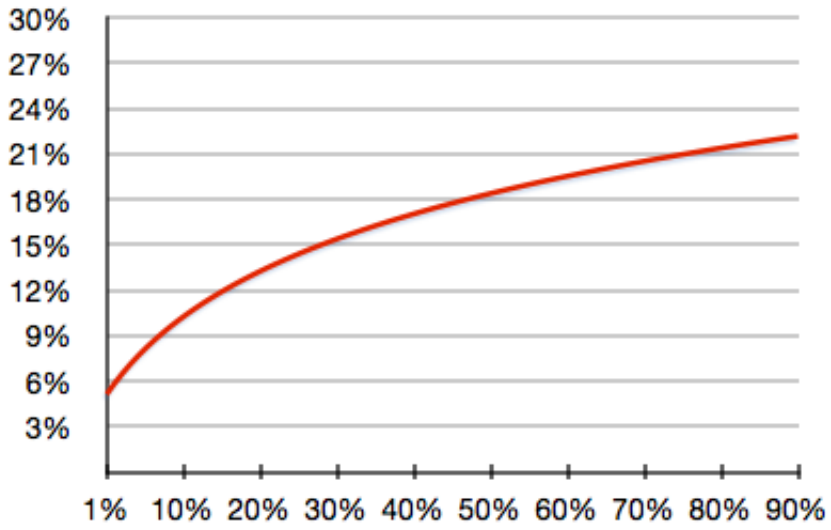


Figura 6.6: Sbilanciamento del numero di nodi contattati nel processo di risoluzione della query

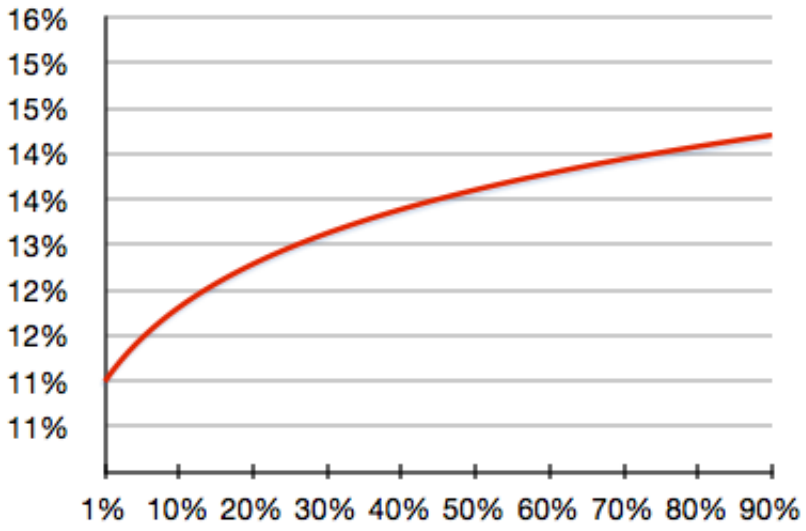


Figura 6.7: Numero medio di nodi contattati

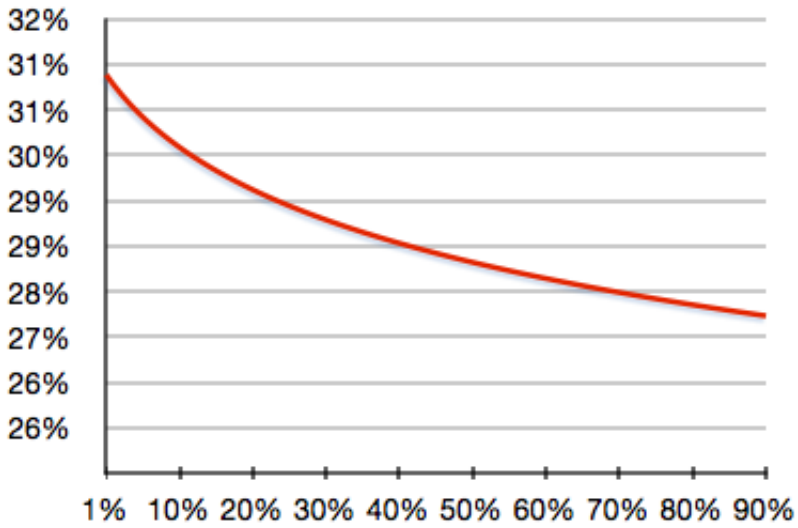


Figura 6.8: Numero di query che raggiungono la radice dell'albero HASP

6.3 Confronto Wavelet vs Bloom Filters

6.3.1 Ambiente di test

PlanetLab

PlanetLab [38] è una piattaforma aperta per lo sviluppo, il deployment e l'accesso a servizi su scala mondiale mediante la rete Internet. La rete dei nodi PlanetLab è composta da 1169 nodi distribuiti in 552 siti in tutto il mondo. Tale infrastruttura è utilizzata per l'esecuzione parallela di job che vengono schedulati quindi su nodi localizzati su scala mondiale. Gli host che fanno parte della rete PlanetLab inviano periodicamente il loro stato ai server, in modo che essi possano controllare lo stato di congestione dei nodi della rete nel tempo e di conseguenza distribuire efficacemente il carico di lavoro sulle varie macchine. Ciascun server ricostruisce, ad intervalli di tempo regolari, a partire dai singoli stati dei nodi, lo stato globale della rete PlanetLab e ne tiene traccia in opportuni file di log. Lo stato di ciascun nodo della rete PlanetLab è definito dai seguenti attributi:

- *FreeSwap:* il totale della memoria di swap libera;
- *CPUUser:* percentuale di utilizzo della cpu da parte di processi in userspace;
- *CPUSys:* percentuale di utilizzo della cpu da parte del kernel;
- *CPUIdle:* Idle time;
- *CPUuse:* tempo di vita dell'host;
- *Load:* numero di processi in esecuzione sul processore negli ultimi 5 minuti;
- *MemPress:* percentuale di capacità di allocare memoria;
- *MemInfo:* percentuale di memoria libera;
- *TXRate:* misura della banda in uscita;
- *RXRate:* misura della banda in entrata;
- *LiveSlices:* numero delle macchine virtuali attive sul nodo.

Definizione dei test

I test sono stati eseguiti per ciascuna delle strutture di digest presentate nel capitolo 4, utilizzando i medesimi file di configurazione per quanto riguarda sia il setting dei valori dei vari attributi sia per quanto riguarda la definizione delle range query multiattributo. I vari test sono stati effettuati in locale tramite l'utilizzo del tool emulatore presente nel framework *OverlayWeaver*. Il numero di nodi virtuali inseriti nella rete HASP è 715, pari al numero di host di cui sono disponibili dati reali in PlanetLab. Tale numero indica inoltre il numero di chiavi derivate che verranno memorizzate sulla rete, in quanto nell'implementazione attuale, ciascun nodo memorizza un'unica chiave. Le chiavi derivate sono dunque generate a partire dal log che rappresenta lo stato delle macchine connesse alla rete PlanetLab in un certo istante di

tempo. Lo spazio delle chiavi derivate utilizzate nei test è pari a $[0; 2^{60} - 1]$. Ciascuna chiave derivata è generata a partire dal valore di 6 attributi (dimensioni) ciascuno dei quali può assumere un valore compreso nell'intervallo $[0; 2^{10} - 1]$. Il valore dei parametri n numero di dimensioni, e k ordine di ciascun attributo, è pari rispettivamente a 6 e 10 questi valori definiscono un possibile scenario tipico. In Figura 6.9 è mostrato un frammento del file di configurazione degli attributi utilizzato nei test.

```
schedule 0 control 0 setdynamic freeSwap 93 - cpuUse 1023 - load 9 - memInfo 856 - TXRate 17 - RXRate 29
schedule 1 control 1 setdynamic freeSwap 159 - cpuUse 931 - load 0 - memInfo 726 - TXRate 20 - RXRate 38
schedule 2 control 2 setdynamic freeSwap 49 - cpuUse 266 - load 2 - memInfo 821 - TXRate 100 - RXRate 94
schedule 3 control 3 setdynamic freeSwap 57 - cpuUse 379 - load 8 - memInfo 904 - TXRate 106 - RXRate 108
schedule 4 control 4 setdynamic freeSwap 345 - cpuUse 235 - load 2 - memInfo 797 - TXRate 61 - RXRate 72
schedule 5 control 5 setdynamic freeSwap 32 - cpuUse 1023 - load 5 - memInfo 952 - TXRate 159 - RXRate 197
schedule 6 control 6 setdynamic freeSwap 125 - cpuUse 164 - load 2 - memInfo 845 - TXRate 61 - RXRate 66
schedule 7 control 7 setdynamic freeSwap 144 - cpuUse 1023 - load 11 - memInfo 892 - TXRate 162 - RXRate 185
schedule 8 control 8 setdynamic freeSwap 108 - cpuUse 1023 - load 9 - memInfo 726 - TXRate 75 - RXRate 60
schedule 9 control 9 setdynamic freeSwap 93 - cpuUse 1023 - load 9 - memInfo 904 - TXRate 30 - RXRate 37
schedule 10 control 10 setdynamic freeSwap 91 - cpuUse 215 - load 6 - memInfo 821 - TXRate 258 - RXRate 226
```

Figura 6.9: Frammento del file di configurazione delle chiavi

Gli attributi utilizzati nei test sono un sottoinsieme di tutti gli attributi analizzati in PlanetLab. I valori degli attributi sono stati estratti da file contenenti snapshot dello stato dei nodi della rete PlanetLab catturati ad intervalli di tempo differenti. I sei attributi scelti sono stati ritenuti i più significativi per la caratterizzazione di uno scenario tipico e sono:

- *freeSwap*
- *cpuUse*
- *load*
- *memInfo*
- *TXRate*
- *RXRate*

Le range query multiattributo sono generate in modo da tenere in considerazione i dati reali stessi, così da adattare l'intervallo di ricerca, per ogni singolo

attributo, alla distribuzione dei suoi valori. Per ciascun attributo si proporziona la distribuzione dei 715 valori sull'intervallo $[0; 1023]$ e per ogni valore intero che l'attributo può assumere, se ne calcola la frequenza. In Figura 6.10 è riportata la distribuzione dei valori dell'attributo memInfo riproporzionata sul range $[0; 1023]$.

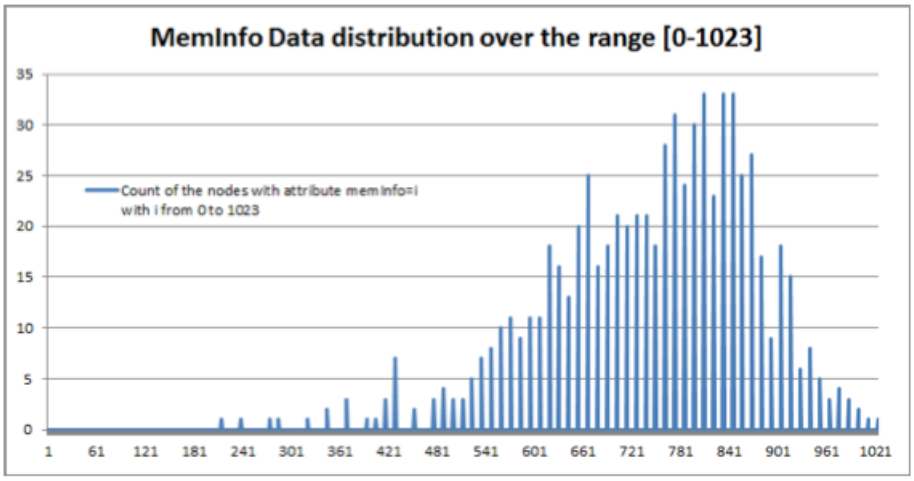


Figura 6.10: Distribuzione dei valori dell'attributo *memInfo* riproporzionata sul range $[0; 1023]$

Una volta calcolate le frequenze dei valori assunti dall'attributo sul range $[0; 1023]$ si genera in maniera casuale, per ciascun attributo, un valore random compreso nell'intervallo $[0.0; 1.0]$.

Il centro dell'intervallo $[a; b]$ che definisce la range query per ogni singolo attributo è dato dal valore x tale che la somma delle frequenze di tutti i valori minori uguali di x è pari al valore random generato casualmente nell'intervallo $[0.0; 1.0]$. I valori a e b che definiscono la range query per ogni attributo sono definiti come:

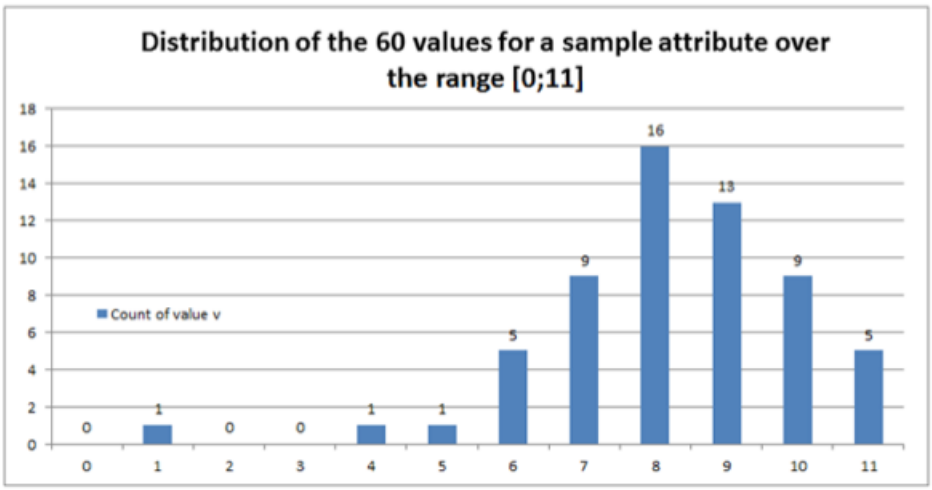
$$\begin{aligned} a = &\quad x - range/2 \\ b = &\quad x + range/2 \end{aligned}$$

dove *range* esprime l'ampiezza massima dell'intervallo della range query. Se il valore delle variabili a o b in seguito a tale calcolo risulta rispettivamente

minore di zero o maggiore del limite massimo dell'intervallo, $2^k - 1$, il valore della variabile a è settato a zero o il valore della variabile b pari al valore massimo. Il valore della variabile *range* è stabilito in base sia alla dimensione dello spazio dei valori di ciascun attributo sia dal numero di attributi che comporranno la range query multiattributo. Se si assegna un valore basso alla variabile *range*, in caso di range query multiattributo con un elevato numero di attributi, la range query di ciascun attributo risulterà essere troppo restrittiva e la range query multiattributo non avrà alcuna chiave che la soddisfi. Ricordiamo infatti che le singole condizioni imposte dalle range query dei vari attributi sono tra loro collegate tramite logica *and*.

Di seguito è riportato un esempio della generazione di una range query guidata dai dati.

Esempio. Le range query per ogni singolo attributo sono state definite in base alla distribuzione dei dati stessi degli attributi. In questo esempio sarà mostrato il procedimento per la definizione di una range query su un attributo a partire dalla sua distribuzione dei dati. Supponiamo di avere un attributo la cui distribuzione dei valori in un range $[0; 11]$ sia nota e mostrata in Figura 6.11.



Frequenze:	0/60	1/60	0/60	0/60	1/60	1/60	5/60	9/60	16/60	13/60	9/60	5/60
	0	0.016	0	0	0.016	0.016	0.083	0.15	0.266	0.216	0.15	0.083

Figura 6.11: Esempio di distribuzione dei 60 valori di un attributo sul range $[0; 11]$

Dalla distribuzione dei 60 valori assunti dall'attributo, è possibile ricavare, per ciascun valore nel range $[0; 11]$, la sua frequenza, ovvero il numero di volte che il valore *h-esimo* si è presentato, con $h \in [0; 11]$. La somma delle frequenze dei vari valori è pari a uno.

Valori:	0	1	2	3	4	5	6	7	8	9	10	11
Frequenza:	0/60	1/60	0/60	0/60	1/60	1/60	5/60	9/60	16/60	13/60	9/60	5/60
	0	0.016	0	0	0.016	0.016	0.083	0.15	0.266	0.216	0.15	0.083
Somma delle frequenze precedenti:	0	0.016	0.016	0.016	0.033	0.05	0.133	0.283	0.55	0.766	0,916	1

Figura 6.12: Frequenze di ciascun valore assunto dall'attributo in Figura 6.11

Una volta calcolate le frequenze di ciascun valore appartenente al range $[0; 11]$ in cui è definito l'attributo, si genera un valore *random* nell'intervallo $[0.0; 1.0]$. In figura 6.12 è mostrata la frequenza relativa a ciascun valore che l'attributo nell'esempio può assumere e la somma delle frequenze dei valori

precedenti il valore *h-esimo*. Supponiamo che nel nostro esempio, la variabile *random* assuma un valore pari a 0.6. Il punto centrale della range query è dato dal valore nel range $[0; 11]$ la cui somma delle frequenze dei valori che lo precedono è pari a 0.6. Nel nostro esempio il punto centrale della range query sarà dunque pari al valore 9 in quanto se alla somma delle frequenze dei valori $[0; 8]$, pari a 0.55, si aggiunge la frequenza del valore 9 (0.216) si ottiene un valore 0.766 maggiore uguale al valore *random*. Una volta ricavato il punto centrale x dell'intervallo $[a; b]$ della range query, si determinano i valori degli estremi a e b tramite il seguente calcolo:

$$\begin{aligned} a = &\quad x - range/2 \\ b = &\quad x + range/2 \end{aligned}$$

Nel nostro esempio assegniamo alla variabile *range* un valore pari a 4. La range query che si otterrà per l'attributo in esempio sarà dunque:

$$7 : 11$$

E' possibile affermare che tale range query è stata definita in maniera guidata dalla distribuzione dei valori dell'attributo stesso poiché il punto centrale della range query è stato scelto tenendo conto delle probabilità che ciascun valore appartenente al range $[0; 11]$ si manifesti per tale attributo.

6.3.2 Validazione Wavelet in PlanetLab

Lo scopo di questa sezione è quello di valutare, al variare della percentuale di pruning, il comportamento della tecnica Wavlet nel *caso reale.* Preso un attributo di PlanetLab, *MemInfo,* analizziamo il livello di memoria occupata e l'errore nella risoluzione di query introdotto così da poter scegliere la percentuale di pruning ottimale.

I parametri di valutazione sono gli stessi adottati nella sezione 6.1. In particolare incrementiamo la percentuale di pruning con step del 10%, ad ogni step viene mostrata:

- per ogni livello la media della dimensione wavelet di tutti i nodi appartenenti a quel livello;
- una media della dimensione wavelet di tutti i nodi presenti nell'albero.
- l'errore introdotto nella risoluzione di range query.

Nella Figura 6.13 è possibile notare l'effetto del pruning sulla dimensione media delle wavelet.

	Pruning									
	0%	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
Livello 1	3	3	3	3	3	3	3	3	3	3
Livello 2	3,109	3,109	3,109	3,109	3,109	3,109	3,109	3,023	3,003	3
Livello 3	3,367	3,367	3,367	3,359	3,351	3,351	3,335	3,070	3,023	3
Livello 4	3,828	3,828	3,812	3,828	3,765	3,734	3,625	3,125	3,046	3
Livello 5	4,843	4,843	4,843	4,781	4,718	4,593	4,3215	3,375	3,125	3
Livello 6	6,937	6,937	6,933	7,325	6,625	6,25	5,25	3,625	3,125	3
Livello 7	11	10,875	10,870	11	10,625	9,125	7	4	3,25	3
Livello 8	19,5	19	19,2	19,5	18,25	15,75	11,5	5,25	3,75	3
Livello 9	36,5	35,5	35,5	41	37	30	23	8	5,5	3
Livello 10	71	67	73	68	66	56	36	10	7	3
Media	16,308	15,7459	16,3634	16,4902	15,6443	13,4912	10,01405	4,6468	3,7822	3

Figura 6.13: Dimensione media wavelet al variare della percentuale di pruning adottata

Di seguito vediamo l'andamento della dimensione media delle wavelet di tutti i nodi dell'albero.

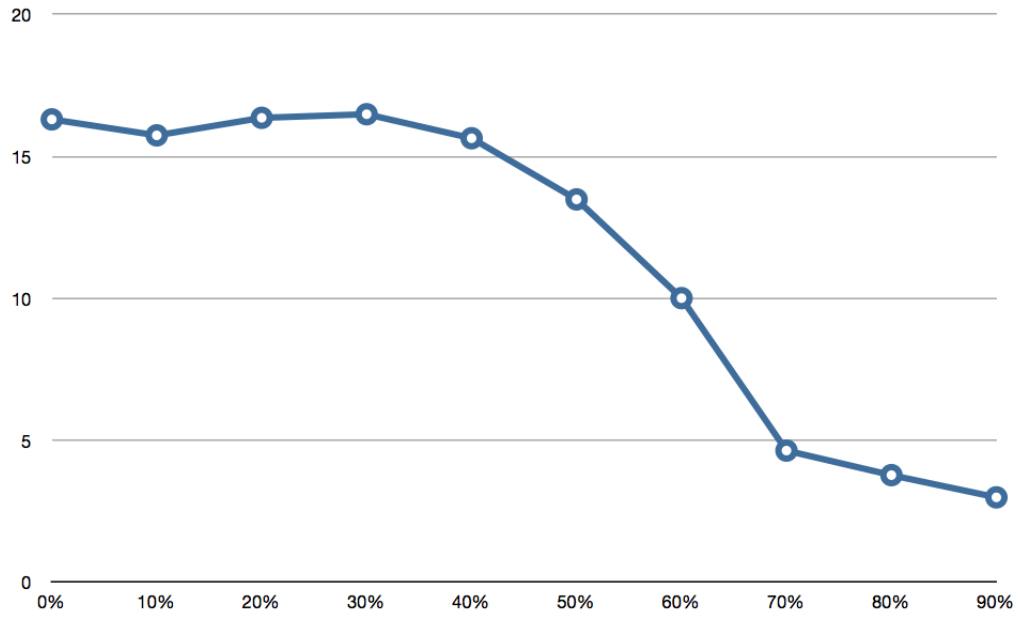


Figura 6.14: Dimensione media wavelet di tutti i nodi dell'albero al variare della percentuale di pruning adottata

Nella Figura 6.15 è possibile notare come per tutti i livelli la media dell'errore globale aumenti all'aumentare della percentuale di pruning. Anche in questo caso l'errore è più rilevante mano a mano che saliamo di livello.

	Pruning									
	0%	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
Livello 1	0	0	0	0	0	0	0	0,001	0,002	0,002
Livello 2	0	0	0	0	0,001	0,001	0,001	0,004	0,008	0,008
Livello 3	0	0	0	0	0,004	0,005	0,007	0,017	0,024	0,024
Livello 4	0	0	0	0	0,019	0,019	0,029	0,049	0,069	0,069
Livello 5	0	0	0,009	0,009	0,079	0,079	0,088	0,128	0,158	0,158
Livello 6	0	0	0	0	0,079	0,079	0,237	0,395	0,395	0,395
Livello 7	0	0	0,039	0,039	0,118	0,197	0,434	0,790	0,790	0,790
Livello 8	0	0	0	0,079	0,474	0,395	0,948	1,660	1,660	1,739
Livello 9	0	0	0	0,158	0,806	0,651	1,360	2,280	2,280	2,5
Livello 10	0	0	0,316	0,632	2,024	2,024	2,213	4,827	5	5
Somma	0	0	0,364	0,917	3,604	3,45	5,317	10,151	10,386	10,685

Figura 6.15: Errore globale nella risoluzione di range query al variare della percentuale di pruning adottata

Di seguito vediamo l'andamento della somma degli errori in tutto l'albero al variare della percentuale di pruning.

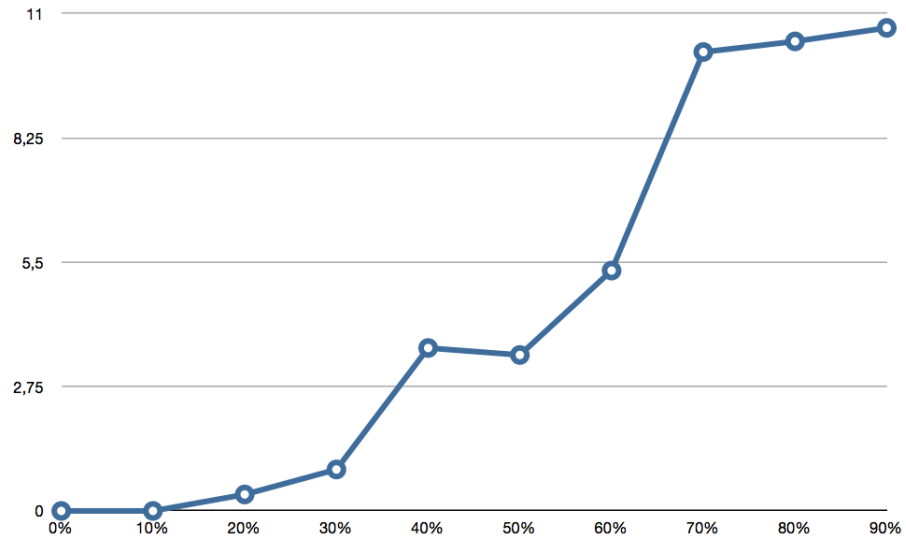


Figura 6.16: Somma degli erriri di tutti i nodi dell'albero al variare della percentuale di pruning adottata

Questi risultati ci mostrano come nel caso reale PlanetLab per l'attributo *memInfo* un buon compromesso sembra essere un pruning del 60%, in quanto produce un buon trade-off tra la memoria occupata e l'errore nel processo di risoluzione query.

6.3.3 Risultati dei test

Le range query multiattributo utilizzate nei test sono dunque composte da $n = 6$ intervalli di ricerca, uno per ogni attributo e da una variabile y la quale esprime il numero di risorse cercate che soddisfino contemporaneamente i sei range $[a; b]$ stabiliti per ogni attributo. In Figura 6.17 è mostrato un frammento del file di configurazione delle range query multiattributo utilizzato nei test.

```
schedule 0 control 0 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 1 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 2 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 3 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 4 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 5 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
timeoffset +10000
schedule 0 control 6 getdynamic freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 5
```

Figura 6.17: Range query multiattributo utilizzate nei test

Per l'esecuzione dei test delle varie strutture di digest sono state definite 10 range query multiattributo differenti e ciascuna di esse è stata eseguita con una strategia *round robin* da ciascuno dei 715 nodi emulati sulla macchina locale. I dati raccolti si basano quindi sull'esecuzione di un totale di 7150 range query. Le dieci range queries utilizzate nei test sono riportate in Figura 6.18.

```
Range Query 1  --> freeSwap 0:220 cpuUse 526:926 load 0:217 memInfo 383:783 TXRate 0:303 RXRate 0:214 y=5
Range Query 2  --> freeSwap 0:278 cpuUse 342:742 load 0:203 memInfo 454:854 TXRate 0:300 RXRate 0:208 y=5
Range Query 3  --> freeSwap 0:251 cpuUse 0:333 load 0:235 memInfo 312:712 TXRate 0:257 RXRate 0:369 y=5
Range Query 4  --> freeSwap 0:231 cpuUse 823:1023 load 0:203 memInfo 466:866 TXRate 0:200 RXRate 0:207 y=5
Range Query 5  --> freeSwap 0:236 cpuUse 0:323 load 0:200 memInfo 585:985 TXRate 0:357 RXRate 0:225 y=5

Range Query 6  --> freeSwap 0:216 cpuUse 383:783 load 0:224 memInfo 645:1023 TXRate 0:336 RXRate 0:340 y=5
Range Query 7  --> freeSwap 0:232 cpuUse 823:1023 load 0:200 memInfo 538:938 TXRate 0:212 RXRate 0:314 y=5
Range Query 8  --> freeSwap 61:461 cpuUse 66:466 load 0:202 memInfo 454:854 TXRate 0:237 RXRate 0:341 y=5
Range Query 9  --> freeSwap 0:340 cpuUse 526:926 load 0:200 memInfo 573:973 TXRate 0:274 RXRate 0:293 y=5
Range Query 10 --> freeSwap 6:406 cpuUse 312:712 load 0:202 memInfo 668:1023 TXRate 0:237 RXRate 0:252 y=5
```

Figura 6.18: Range queries multiattributo utilizzate per i test

Le misure analizzate nei test sono:

1. sovrastima media del numero di risultati ricevuti dal processo di risoluzione di una query;
2. sbilanciamento medio del numero di nodi contattati: differenza tra il numero massimo e numero minimo di nodi contattati nella risoluzione della stessa range query eseguita da nodi diversi;
3. percentuale media di query che raggiungono la radice dell'albero HASP nel processo di risoluzione di una query;

4. percentuale media del numero di nodi contattati rispetto al numero di nodi totali della rete durante il processo di risoluzione di una query.

Il primo test fornisce una misura della *query satisfaction*, ovvero del numero medio di *match* restituiti per ciascuna query di test. E' un dato di fatto che un'informazione di digest può sottostimare il numero di match di una query che sono presenti in un sottoalbero. Quando ciò accade, il numero di risultati restituiti al nodo che ha effettuato la query è maggiore del numero k richiesto.

Il numero di nodi coinvolti nel processo di risoluzione di una query può variare a seconda di quale nodo dell'albero invochi la query. Il secondo test misura lo sbilanciamento medio del numero di nodi contattati nell'esecuzione della stessa query da parte di nodi diversi.

Il terzo test si propone di investigare il numero medio di query che raggiungono il peer che gestisce la radice dell'albero di aggregazione. Un'informazione di digest è utile per riassumere in un nodo l'informazione contenuta nel sotto-albero radicato in tale nodo. Una strategia di aggregazione è tanto migliore quanto maggiore è l'accuratezza delle informazioni contenute nei nodi riguardo il suo sotto-albero. Tanto minore è il numero di nodi contattati rispetto al numero totale di nodi, durante il processo di risoluzione di una query, tanto migliore è la strategia di aggregazione utilizzata.

Risultati del confronto tra Wavelet e Bloom Filters

Il grafico in Figura 6.19 mostra la sovrastima media del numero di risorse trovate rispetto al valore y di risorse richieste. In questo contesto quando parliamo di media ci riferiamo al valore medio calcolato dall'esecuzione della stessa query da parte dei 715 nodi della rete. Possiamo notare come per 9 queries su 10 l'utilizzo della strategia di digest Wavelet conduca ad una percentuale media di sovrastima minore rispetto all'utilizzo dei Bloom Filters. Questo risultato sta ad indicare come le Wavelet forniscano un'approssimazione più precisa delle informazioni contenute nei sotto-alberi dei nodi in HASP.

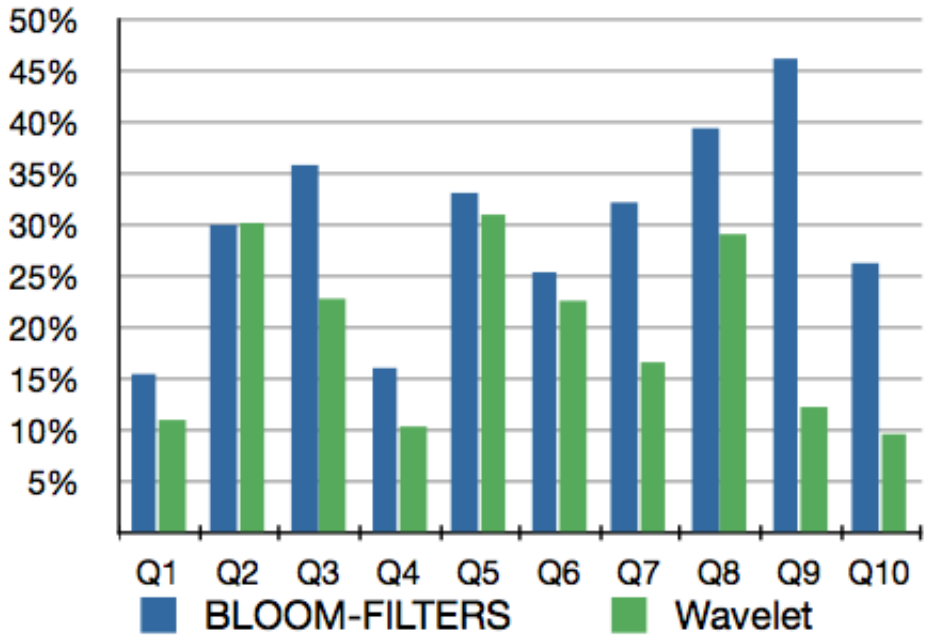


Figura 6.19: Sovrastima media del numero di risultati ricevuti rispetto al valore y richiesto

Il grafico in Figura 6.20 mostra lo sbilanciamento medio tra il massimo numero di nodi contattati ed il minimo numero di nodi contattati durante la risoluzione di una query. Possiamo notare come per 9 queries su 10 lo sbilanciamento medio del numero di nodi contattati sia minore utilizzando la strategia di aggregazione Wavelet. Questo sta ad indicare come l'utilizzo delle wavelet come strategia di digest, rispetto ai Bloom Filters, conduca ad un minor traffico nella rete nel processo di risoluzione di una query.

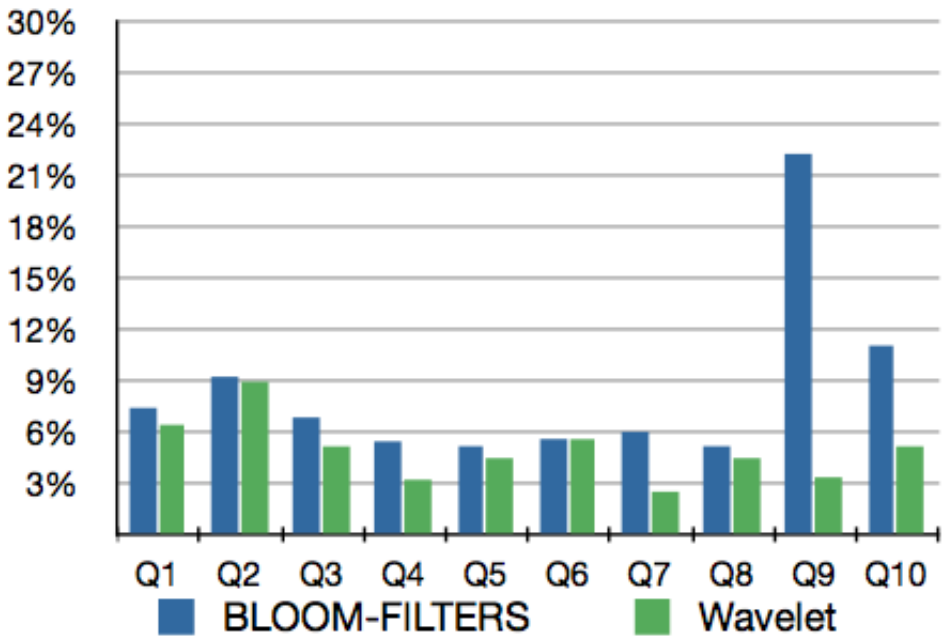


Figura 6.20: Sbilanciamento del numero di nodi contattati nel processo di risoluzione della query

Il grafico in Figura 6.21 mostra in numero medio di query che durante il processo di risoluzione raggiungono la radice dell'albero HASP. E' possibile notare come per tutte le 10 queries entrambe le strategie di digest, wavelet e Bloom Filters, ottengano risultati identici.

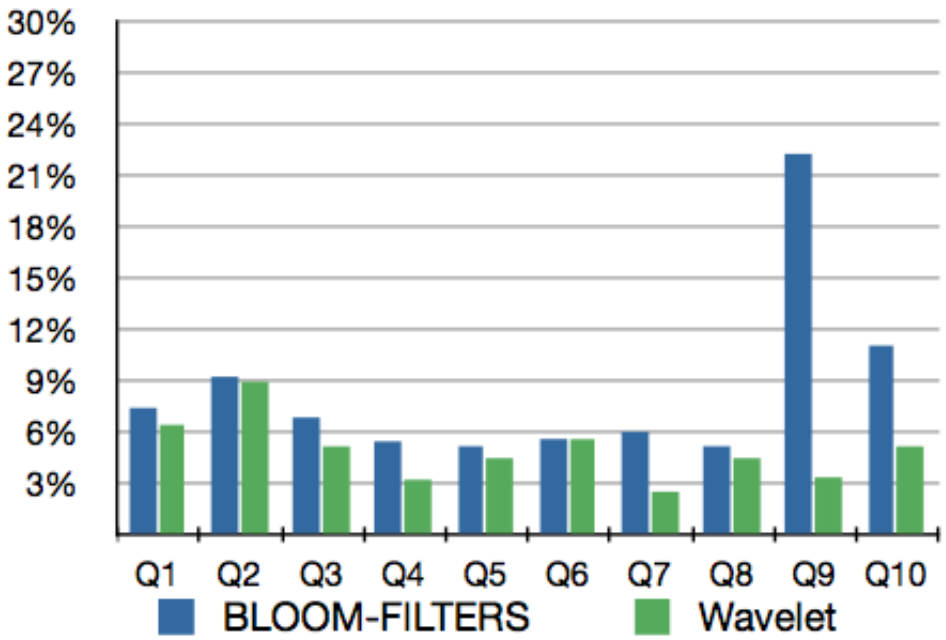


Figura 6.21: Numero di query che raggiungono la radice dell'albero HASP

Il grafico in Figura 6.22 mostra il numero medio di nodi contattati rispetto al numero di nodi totali della rete durante il processo di risoluzione di una query. Possiamo notare come, per tutte le queries, utilizzando la tecnica di digest wavelet il numero medio di nodi contattati sia minore rispetto alla tecnica Bloom Filters. Questo risultato sta ad indicare come la bontà di approssimazione appartenente alle wavelet si traduca nella generazione di minor traffico di rete all'interno di HASP.

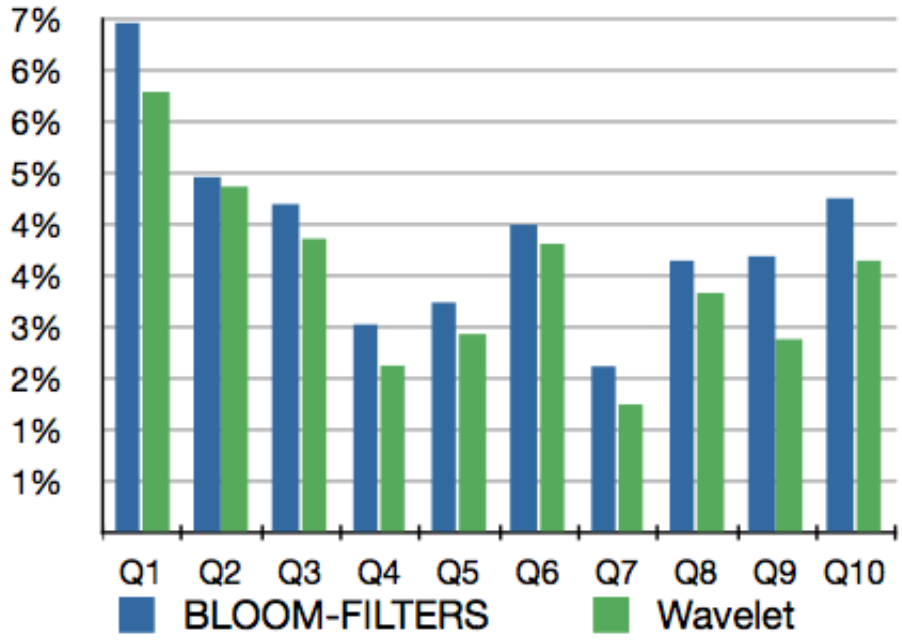


Figura 6.22: Numero medio di nodi contattati

Capitolo 7

Conclusioni

Il contributo di questa tesi è stata la definizione di nuove tecniche di digest da integrare in HASP, un sistema per la ricerca di risorse in ambienti P2P, basato sulla definizione di alberi di aggregazione distribuiti. L'introduzione di nuove tecniche di digest è stata la conseguenza di un'attenta analisi delle proposte attualmente presenti in letteratura. In particolare sono state scelte ed inserite due nuove tecniche ispirate ai Bloom Filters ed alle Wavelet.

L'integrazione delle Wavelet ha arricchito HASP di una tecnica di digest molto sofisticata. Per integrare le Wavelet come informazione di digest nell'albero di aggregazione definito in HASP è stato necessario definire un nuovo algoritmo per il merge di due Wavelet. Inoltre la grande dimensione dello spazio dei valori delle chiavi derivate, ottenute mediante la tecnica delle curve space filling, ha richiesto la definizione di una rappresentazione ottimizzata per la distribuzione dei dati, tale ottimizzazione ha portato all'introduzione di un nuovo algoritmo per il calcolo delle Wavelet.

L'integrazione dei Bloom Filters ha permesso di inserire in HASP una tecnica di digest che utilizzi con una struttura dati probabilistica compatta. I Bloom Filters sono stati implementati in modo da poter avere una probabilità di falsi positivi fissata, di conseguenza è stato svolto un lavoro di tuning per individuare il giusto trade-off tra la memoria utilizzata e la loro affidabilità.

Sono stati effettuati un insieme di esperimenti sia su dataset sintetici che reali. In particolare sono stati utilizzati dati reali riguardanti le risorse presenti in PlanetLab. I risultati hanno messo in evidenza l'utilità delle wavelet come tecnica di aggregazione dei casi, in cui e' possibile individuare un trade-off tra lo spazio necessario per rappresentare i dati aggregati e l'approssimazione introdotta per la loro rappresentazione. Dal confronto tra Wavelet e Bloom Filters è emerso come la prima tecnica sia preferibile sia perchè riduce la quantità di nodi dell'albero di aggregazione visitati e consente di evitare di dover risalire l'albero di aggregazione fino alla radice.

7.1 Sviluppi futuri

Il lavoro di questa tesi può essere esteso in diverse direzioni. Ad esempio, è possibile valutare la possibilità di utilizzare wavelet multidimensionali come alternativa alla tecnica delle curve space filling. Inoltre, un'analisi più completa della tecnica proposta può riguardare la sperimentazione su altri dataset reali, diversi da PlanetLab.

Elenco delle figure

2.1 Classificazione query 11

2.2 Topologia reti P2P 13

2.3 Organizzazione delle strutture dati P2P 15

2.4 Squid: Funzione di Hilbert 17

2.5 Squid clusters 17

2.6 Raffinazione dei clusters in Squid 18

2.7 Routing table di un nodo in Baton 19

2.8 Indici distribuiti in Baton 20

2.9 Esempilo di bilanciamento del cairco dei nodi in Baton 21

2.10 Matrice di bilanciamento del carico (LBM) per (a_i, v_i) 23

2.11 Esempio struttura RST 24

2.12 Determinazione della banda 25

2.13 Architettura CONE 27

2.14 Casi di massimo sbilanciamento dell'albero Cone 29

2.15 Caso generico del Data Traffic in Cone 30

3.1 Trasformata di Haar su segnale discreto 40

3.2 (a) Segnale originale. (b) Trasformata di Haar di secondo livello. (c) Energia del segnale originale. (d) Energia del segnale risultante dalla Trasformata di Haar di secondo livello. 46

3.3 Error tree 59
3.4 Error tree 62
3.5 Bloom filter operazione di add 64
3.6 Bloom Filter operazione di look up 64
3.7 Counting Bloom Filter operazione di add 65
3.8 Probabilità falsi positivi all'aumentare dei bit assegnati ad ogni elemento 67
3.9 Probabilità falsi positivi al variare degli elementi dell'insieme . 68

4.1 Overlay XCone-DHT 73
4.2 Join di un peer nella rete XCone 75
4.3 Localizzazione *LCA* 76
4.4 Fase di *trickling* in XCone 77

5.1 Architettura Overlay Weaver 100
5.2 Tipologie di routing in Overlay Weaver 100
5.3 Esempio di scenario interpretabile dall'emulatore 102
5.4 Esempio di visualizzazione grafica della rete 102
5.5 Scenario sezione 1, definizione ed inizializzazione dei nodi del sistema 103
5.6 Scenario sezione 2, specifica ed assegnazione delle risorse ai nodi 104
5.7 Scenario sezione 3, *join* dei nodi sulla rete 104
5.8 Scenario sezione 4, definizione delle range query 105

6.1 Dimensione media wavelet al variare della percentuale di pruning adottata 115
6.2 Dimensione media wavelet di tutti i nodi dell'albero al variare della percentuale di pruning adottata 115

6.3 Errore globale nella risoluzione di range query al variare della percentuale di pruning adottata 116

6.4 Somma degli errori di tutti i nodi dell'albero al variare della percentuale di pruning adottata 116

6.5 Sovrastima media del numero di risultati ricevuti rispetto al valore y richiesto 118

6.6 Sbilanciamento del numero di nodi contattati nel processo di risoluzione della query 119

6.7 Numero medio di nodi contattati 119

6.8 Numero di query che raggiungono la radice dell'albero HASP . 120

6.9 Frammento del file di configurazione delle chiavi 122

6.10 Distribuzione dei valori dell'attributo *memInfo* riproporzionata sul range $[0; 1023]$ 123

6.11 Esempio di distribuzione dei 60 valori di un attributo sul range $[0; 11]$ 125

6.12 Frequenze di ciascun valore assunto dall'attributo in Figura 6.11 125

6.13 Dimensione media wavelet al variare della percentuale di pruning adottata 127

6.14 Dimensione media wavelet di tutti i nodi dell'albero al variare della percentuale di pruning adottata 128

6.15 Errore globale nella risoluzione di range query al variare della percentuale di pruning adottata 128

6.16 Somma degli erriri di tutti i nodi dell'albero al variare della percentuale di pruning adottata 129

6.17 Range query multiattributo utilizzate nei test 130

6.18 Range queries multiattributo utilizzate per i test 130

6.19 Sovrastima media del numero di risultati ricevuti rispetto al valore y richiesto 132

6.20 Sbilanciamento del numero di nodi contattati nel processo di risoluzione della query 133

6.21 Numero di query che raggiungono la radice dell'albero HASP 134

6.22 Numero medio di nodi contattati 135

Bibliografia

[1] R.Steinmetz and K. Wehrle. Peer to Peer Systems and Applications. LNCS. 3485, Springer Verlag, 2005.

[2] C. Kesselman and I. Foster. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, November 1998.

[3] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incen- tives in p2p systems. Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings for the Third International Conference on, pages 48-56, September 2003.

[4] R. Matei, A. Iamnitchi, and P. Foster. Mapping the gnutella network. Internet Computing, IEEE, 6(1):50-57, 2002.

[5] Napster. http://www.napster.com/, 2006.

[6] J. A. Pouwelse, P. Garbacki, D. H. Epema, and H. J. Sips. The bittorrent P2P file-sharing system: Measurements and analysis. Proceedings of the 54th Inter- national Workshop on Peer-to-Peer Systems (IPTPS'05), pages 205-216. 2005, Ithaca, USA, 2005.

[7] Li Gong. JXTA: a network programming environment. Internet Computing, IEEE, 5(3):88-95, 2001.

[8] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa net- work. Proceedings of the The Third IEEE Workshop on Internet Applications (WIAPP '03), pages 112-120, June 2003.

[9] C. Kesselman and I. Foster. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, November 1998.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan. CHORD: a scalable peer-to-peer lookup protocol for inter- net applications. IEEE/ACM Transactions on Networking (TON), 11(1):17-32, February 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR00-010, Berkeley, CA, 2000.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, 2001.

[13] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubia- towicz. Tapestry: a resilient global-scale overlay for service deployment. Selected Areas in Communications, IEEE Journal on, 22(1):41-53, 2004.

[14] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. Electronic Proceedings for the 1st Interna- tional Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club - Cambridge, March 2002.

[15] M. Cai, M. Frank, J. Chen and P. Szekely. MAAN: a multi-attribute addressable network for grid information services. Grid Computing, 2003. Proceedings for the Fourth International Workshop on, pp. 184-191, November 2003.

[16] C. Schmidt and M. Parashar. Analyzing the search characteristics of space filling curve-based indexing within the squid p2p data discovery system. Technical report, Rutgers University, December 2004.

[17] H. V. Jagadish, Beng C. Ooi, and Quang H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In Proceedings of the 31st Internationational Confe- rence on Very Large Data Bases (VLDB '05), pp. 661-672. Trondheim - Norway, September 2005.

[18] Jun Gao. A Distributed and Scalable Peer-to-Peer Content Discovery System Supporting Complex Queries. PhD thesis, Computer Science, Carnegie Mellon University, October 2004.

[19] G. Varghese, R. Bhagwan, P. Mahadevan and G. M. Voelker. Cone: A distri- buted heap-based approach to resource selection. Technical report, University of California, 2004.

[20] M. Marzolla, M. Mordacchini, and S. Orlando. Tree vector indexes: efficient range queries for dynamic content on peer-to-peer networks. In Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on, pages 8 pp.+, 2006.

[21] N. Shrivastava, C. Buragohain, D. Agrawal and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 239-249, New York, NY, USA, 2004. ACM.

[22] Haar, Alfréd (1910), "Zur Theorie der orthogonalen Funktionensysteme", Mathematische Annalen 69 (3): 331–371

[23] Y. Matias, J.S. Vitter, and M. Wang. "Dynamic Maintenance of Wavelet-based Histograms". VLDB 2000.

[24] J.S. Vitter, and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. SIGMOD '99 Proceedings of the 1999 ACM SIGMOD international conference on Management of data, pp. 193-204.

[25] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. Proceedings of the 1998 ACM SIGMOD interna-

tional conference on Management of data, p.448-459, June 01-04, 1998, Seattle, Washington, United States.

[26] A Primer on Wavelets and their Scientidic Applications, James Walker, Chapmann & Hall.

[27] Burton H. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors, in Communications of the ACM, Volume 13, Number 7, 1970.

[28] Fay Chang et al., Bigtable: A Distributed Storage System for Structured Data, in OSDI'06: Seventh Symposium on Operating System Design and Implementation, 2006.

[29] A. Rousskov and D. Wessels. Cache digests. Computer Networks and ISDN Systems, 30(22-23): 2155-2168, 1998.

[30] N. Shrivastava, C. Buragohain, D. Agrawal and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 239-249, New York, NY, USA, 2004. ACM.

[31] D. Carfì, M. Coppola, D. Laforenza and L. Ricci. DDT: A Distributed Data Structure for the Support of P2P Range Query. Proceedings of the IEEE Col- laborateCom 2009, 5th International Conference on Collaborative Computing, Networking and Applications, Washington, November, 2009.

[32] M. Parchi. Analisi e sperimentazione di curve space-filling per range query multiattributo in sistemi P2P. Tesi di Laurea, Corso di Laurea Specialistica in Tecnologie Informatiche, Dicembre 2010.

[33] J. K. Lawder. The Application of Space-filling Curves to the Storage and Retrieval of Multi-dimensional Data. PhD Thesis, Birkbeck College - University of London, 1999.

[34] H. Sagan. Space-Filling Curves. Universitext series. Springer-Verlag, Heidelberg, 1994.

[35] I. H. Witten and B. Wyvill. On the generation and use of space-filling curves. Software - Practice and Experience, 13(6):51 9-525, June 1983.

[36] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. Computer Communications, (2):402–412, February. Framework available at http://overlayweaver.sourceforge.net.

[37] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common API for structured peer-to-peer overlays. Peer-to-Peer Systems II, Second International Workshop, (IPTPS 2003), Berkeley - USA, February 2003.

[38] PlanetLab: An open platform for developing, deploying, and accessing planetary- scale services. Website: http://www.planet-lab.org/